programmer's journal

The Resource Journal for IBM PC Programmers

"I own the software."

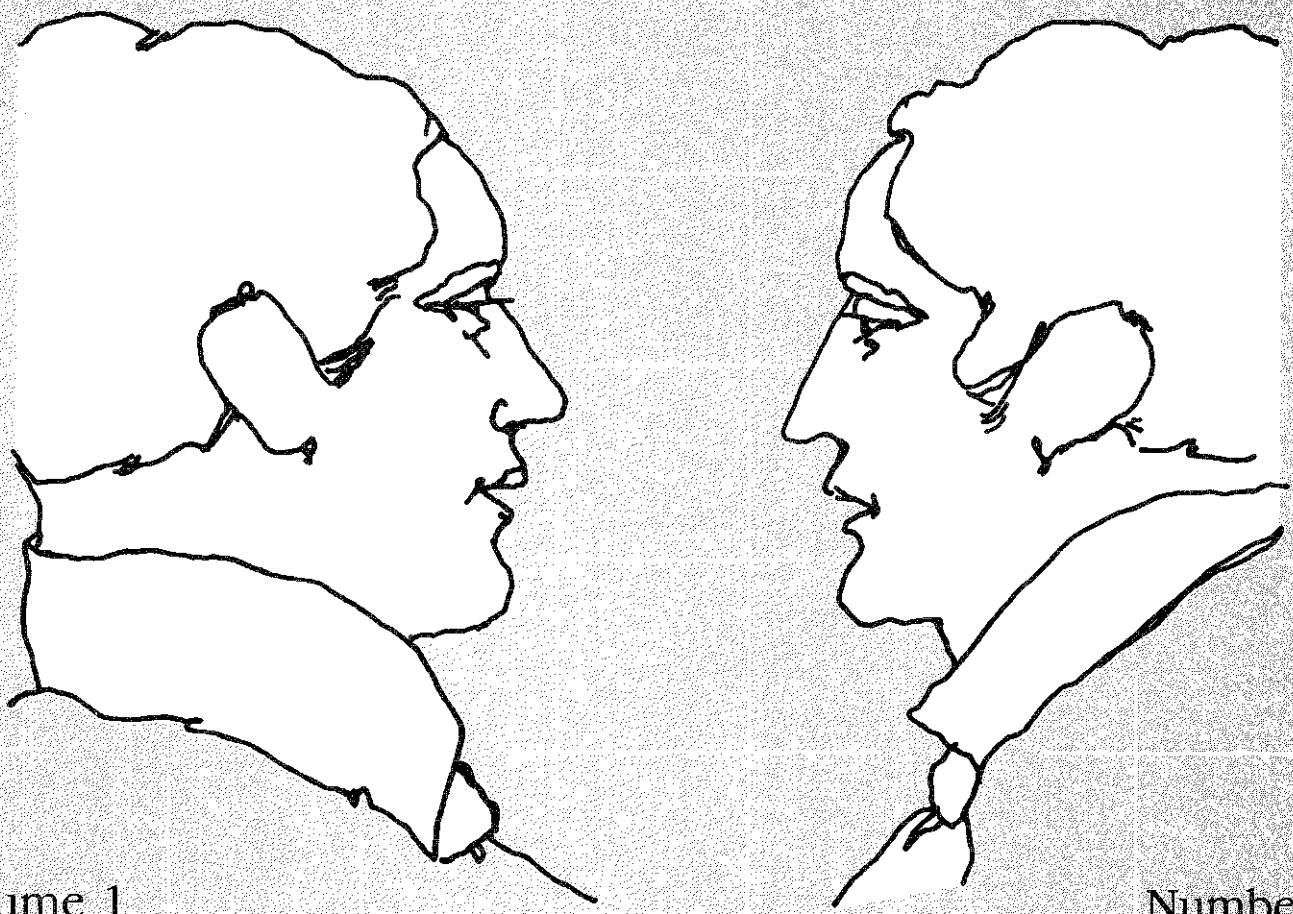
"No, it belongs to me!"

"Look, I wrote it."

"But I paid for the development."

"You can't be serious."

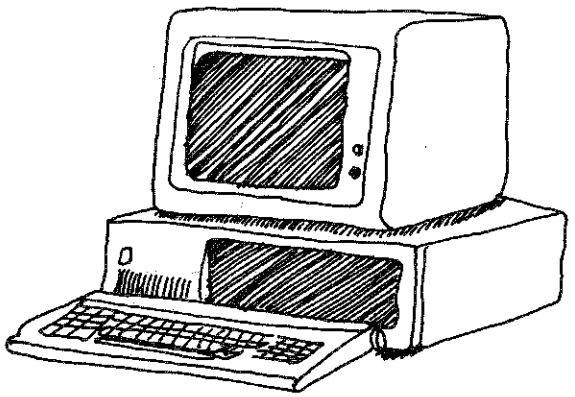
"I've already sold the marketing rights."



Volume 1 Number 4

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



CONTENTS

HOUSEKEEPING 2

LETTERS 3

PJ PROFILE/Interview with Robert Dunaway 5
A look at Software Law

DOS 2.0: IMPRESSIONS/Morton F. Kaplon 8
Write your own Device Drivers

ASM: BEEP AND BEEP STAY RESIDENT/Art Zachai 11
Make your routines resident

C: JOYSTICK INTERFACE ROUTINES/Bill Hailey 14
Speed up your C86 programs

PASCAL: DIRECTORY ACCESS/Walter Rauser 15
Pascal Tutorial Disk Excerpts

ASM: INTERLOCKING MACROS:WHILE-WEND/Dan Rollins 18
Using your Macro Assembler to the fullest

BASIC: STRUCTURED PROGRAMMING (Part I)/Joe Rayhawk 20
Logical approaches to BASIC coding

UTILITIES: DISK EXPLORER/Richard Larratt 24
The last disk utility you may ever need

TOOLS: DEFAULT VALUES & VARIABLE NAMES/Greg Estes 25
Abuser friendly techniques

PJ FORUM: HOT SPOTS/Neil Bennett 27
Look closely at your trouble spots

SOFTWARE REVIEWS
HOSTCOMM/Jim Bumpas 29
THE PROGRAMMER/Ronald Moak 30
PROOFREADER/David Whitman 34

dBASE II: REPORT TIME/Frankie Detesto 36

PJ LIBRARY 37

MISC 39

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSEKEEPING

HOUSEKEEPING

I have to put to rest Ron Cain's article about writing a Small-C compiler for an 8080 based computer. It isn't the actual code that keeps resurrecting itself, but the method of "bootstrapping" yourself up from one version of a compiler to the next. The concept has always left me a little unsettled, like standing on top of 3 chairs and needing just one more to reach the ceiling so you take out the bottom chair and put it on top. It sure won't work with chairs, but it does seem to function quite normally with compilers.

Maybe if we go through the process it will finally seem 'OK" to me and then YOU can worry about.

Let's assume we use our IBM PC and we have the Macro Assembler and a BASIC interpreter running on it. We decide we want to write a C compiler. The solution revolves around our ability to write a FIRST simple C compiler using BASIC which reads C source and creates some 8088 ASM code (a sort of ASM code generator.) Then we use the Macro Assembler to transform it into a final executable module.

We stay up late for several weeks and soon the first version of our C compiler is complete. We have written it in BASIC. The next step is to run a C source code text file through it to test it. What kind of program shall we write in C to try out our first version? It could be most anything. A text editor or a mailing list program would certainly do. But we decide to write a C compiler since we are quite familiar with it (we just did it in BASIC.)

So we are now going to test our BASIC version of a C compiler on a C text file (that just happens to be another compiler.) The BASIC program reads in the C text file and writes out ASM code. It messes up a few times and we have to tweek it to produce some nice looking mnemonic code. Finally the ASM code produced by our BASIC C compiler appears fairly normal. The ASM code is then run through the Macro Assembler and we have a (hopefully) runable version from the original C source file.

Might as well test our new executable version. This "C compiler" that we created (via the BASIC interpreter and MASM) is ready to chop away on some C code. What kind of C source code shall we use to test our running version? You guessed it. Why not feed it its own source code file? Use our original C source to test our running C compiler.

Do you see why it gets a little muddy? Once the first executable module is created we can continue development on our C compiler by expanding the C source code file. If at any point a bug was discovered, and assuming you kept ALL versions as you bootstrapped yourself up, then you could fix a bug several levels down and then recompile back up to the latest version.

Perhaps it would be easier to think about buying a copy of IBM's Pascal compiler and then sitting down to write your OWN Pascal compiler using theirs. The mental gymnastics begin when your first clean compile comes through and you want to test it on some Pascal code (and you feed it your Pascal source file.)

I feel better now about the whole concept. If it bothers you then maybe explaining it to a fellow programmer would help.

Next time we will discuss using a BASIC program generator to write a BASIC program generator. Just kidding of course.

Your fellow programmers that have contributed to this issue are M.F. Kaplon, Dan Rollins, Art Zachai, Ron Moak, Neil Bennett, David Whitman, Joe Rayhawk, Bill Hailey, Richard Larratt, Jim Bumpas, Walter H. Rauser, Frank Greco and Joseph McDermott.

— *Greg Estes*

PROGRAMMER'S JOURNAL

Editor/Publisher: Greg Estes
Associate Editor: Frankie Detesto
Contributing Editor: Greg Roberts
Art Director: Julia King
Production: Cassandra Estes
Copy Editing: Renee Anderson

Editorial and Business Office:
2765 Potter St.
Eugene, OR 97405

Mailing Address:
P.O. Box 5070, Eugene, OR 97405

Programmer's Journal. Volume 1, Number 4.

Copyright (C) 1983 by Programmer's Journal

Published bi-monthly.

Postmaster send address changes to:
Programmer's Journal
2765 Potter St.
Eugene, OR 97405
ph. (503) 687-0398

Subscriptions: Bi-monthly rates are $24 U.S. ($30 in Canada, and $40 elsewhere.)

IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York
dBASE II is a trademark of Ashton-Tate.

Contributing Writers:
Joe Rayhawk
Arthur Zachai
Morton F. Kaplon
Neil Bennett
Dan Rollins
Ron Moak
Jim Bumpas
David Whitmen
Bill Hailey
Richard Larratt
Walter H. Rauser
Frank Greco
Joseph McDermott

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

GRAPHICS SETS

A few comments and questions regarding your informative magazine.

1. Re: User Defined Character Sets (March/April '83.) I am a little confused about Mark's program segment (bottom right page 23.) If you know the address (absolute address) then,

```
OFFSET = BOTTOM MOD 65536
SEGMENT = BOTTOM/65536
```

Unless I missed something, isn't this a more straight forward method? Why did Mark multiply INT(BOTTOM/65536) by 4096? Another question, the postscript says to use

```
OFFSET = VARPTR(TBL#(1))
```

Why is the VARPTR$ used instead of VARPTR ? From experimenting with these functions, they return different values for the same variable. Perhaps I'm confused regarding them.

Also an item to remember; the BASIC manual says that all scalar variables should be declared prior to using VARPTR (especially as a vectored interrupt!) because every new scalar causes the addresses of all arrays to be changed and hence would destroy any chance of retrieving the character set.

2. Since we're talking about new characters, I've written several arcade style games using the monochrome monitor and had surprisingly good results. I recently attempted to convert them so that they could run on the color graphics monitor (I used Peeks and Pokes.) While the conversion was fairly trivial (moving DEF SEG = &HB800 instead of DEF SEG = &HB000) something unusual occurred. I received a lot of "snow" on the monitor (color.)

After talking with several people, it is apparently the result of Poking when "retrace" is occurring (?), or not occurring (?). What does that mean and how do I circumvent it? Some of my games are in compiled BASIC and are running as fast as they can. Checking for a blanking period from the monitor would severely limit the programs and would call for a redesign.

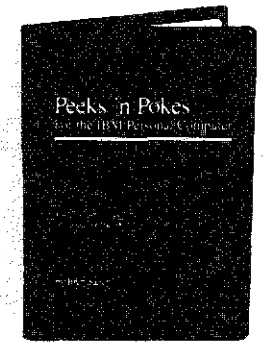
3. Accessing BIOS from C86 by James Randall was informative. A note: BASIC's LINE command SHOULD be much faster than James' program because most graphic commands (not all) write directly to the video memory and bypass BIOS. Unfortunately IBM's BIOS (graphics calls especially) is sub par in terms of functionality and speed. Many software people are totally bypassing the BIOS and writing directly to RAM. This is understandable yet regrettable because it renders the code unportable.

4. Art Zachai's Assembler article was extremely informative. Need more of the same.

5. It might be interesting to look at the IBM PC version of Modula II by Valition Systems. From what I've read, it appears to be an up and coming language.

Frank Greco
Kearny, NJ

Peeks 'n Pokes For the IBM PC puts you in control of your PC!



Are you writing programs in BASIC or Pascal? This unique product has a disk with 58 programs and a 38-page manual that helps you get 'underneath the covers' of the PC. Learn how to use PEEK, POKE, INP, OUT, and DOS/BIOS function calls to do what you want, fast! Do you want to perform functions not available from BASIC or Pascal? It's all explained in the manual and demonstrated in the sample programs. You can increase the capabilities of your programs while making them even easier to use.

The easy to learn techniques save you time and increase your productivity.

Peeks 'n Pokes shows you how to:

- Access the system's configuration
- Unprotect BASIC programs
- Scroll part or all of the screen
- Access the file directory
- Logically swap printers
- Read and change the keyboard
- Find more Peeks 'n Pokes
- And much more...

All this for only $30.00 plus $2.50 shipping. MasterCard and VISA accepted.

Data Base Decisions • 14 Bonnie Lane • Atlanta, GA 30328 • 404/256-3860

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRECISION COMPUTATION

Neil Bennett's letter on numeric precision stimulated me to write this reply. I have now seen more or less the same problem discussed in a dozen microcomputer journals. Actually, the numeric precision problem disappears if a few simple rules are followed. Unfortunately, these guidelines do not appear in any of Microsoft's (or IBM's) manuals.

The rules have a solid basis in number theory and MBASIC's architecture, but a letter is no place to be long-winded. So here are the rules:

1. Always use double precision for amounts of money and other quantities if exact values are required. You MUST be consistent from the beginning to the end of your program. NO EXCEPTIONS. If you want double precision results, ALL values entering your calculations MUST be double. Single precision is —expletive deleted—.

The following rules implement Rule #1.

2. For assignment statements and formulas, ALL numeric quantities must be followed by the pound sign which signifies double.

Examples:

```
.....Good.....
NUM# = 1.19#
NUM# = NUM#*3.1#

.....Bad.....-
NUM# = 1.19
NUM# = NUM#*3.1
```

3. For program output, ALWAYS use PRINT USING with double precision. Anyone who writes a line like "PRINT NUM#" deserves what he gets.

```
.....Good.....
PRINT USING "####.##" ,NUM#

.....Bad.....-
PRINT NUM#
```

4. Program input like "INPUT NUM#" will work correctly but serious programmers avoid the INPUT statement. They use INKEY with character validation to construct a numeric string. This must be converted to double precision. We invoke rule #2.

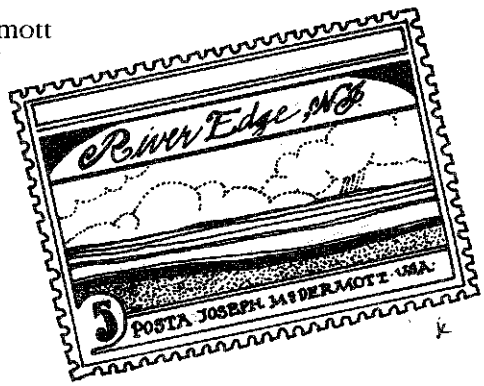
```
.....Good.....
NUM# = VAL(NUM$ + " #" )

.....Bad.....-
NUM# = VAL(NUM$)
```

Remember that these are RULES, not suggestions. Use them without exception in all business and accounting programs. They eliminate all of the problems I have so far encountered.

Finally, I would like to answer one specific point made by Dr. Bennett. It is true that 1.19 cannot be represented exactly by binary floating point. But if you follow the rules presented here it will be accurate to 15 decimal digits. That's enough to cover the National Debt to the penny - at least for a few more years !! Of course, if you are a programming consultant for Darth Vader and the Galactic Empire, you will have to round to the nearest thousand.

Keep up the good work with your interesting magazine.

Joseph McDermott
River Edge, NJ



NOT GOOD ENOUGH

I hope the magazine improves.

Ken Cleveland
Burke, VA

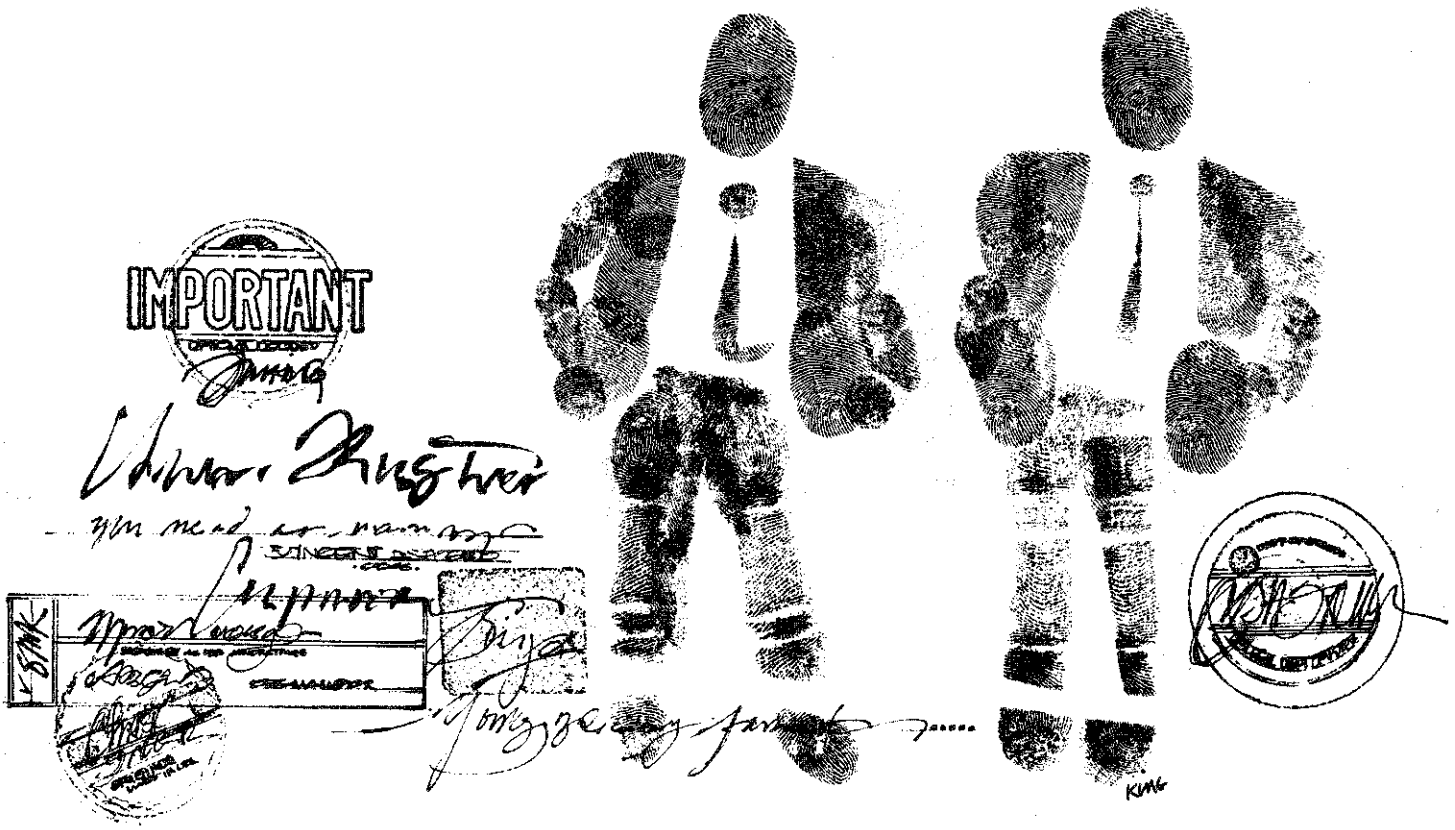
DON'T GO GLOSSY

I have just read Volume 1 of PJ and would like to subscribe. As a fellow programmer of the IBM PC (my partner and I have been working on an accounting package for attorneys for the last 11 months,) I enjoyed the fact that the magazine is tailored specifically to programmers. Also, as a "cottage industry" programmer, (we both have full time jobs in other areas,) I like the relaxed format of the magazine. Please don't go to glossy covers and pages!

Bill Gibbs
Tempe, AZ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PROFILE



SOFTWARE LAW

an interview with Robert Dunaway, a partner in the law firm of Remer, Remer, & Dunaway

PJ: Are there any especially new developments in the software law field?

DUNAWAY: Probably the hottest area right now is with the software rental firms. The larger software publishing companies have been wondering what to do with them for some time.

One of the biggest publishing houses had been talking about the rental problem for some time. On June 17 of this year Micropro filed one of the first (if not THE first) actions against a software rental company. This was done here in California. The complaint was for copyright infringement, breach of copyright license, unfair competition and unjust enrichment.

So they are essentially asking for an injunction, damages, and punitive damages.

PJ: Does a resale house rather than a rental company actually sign a contract with a software publishing house?

DUNAWAY: The typical resale house does sign a fairly extensive distributor agreement which includes mention of a single terminal, single user installation.

PJ: And I suppose the rental house just bypasses the agreement completely.

DUNAWAY: Yes, and that is the problem. All the rental house needs is one copy of a product to go into business.

PJ: What technical grounds are the software houses going to use to combat the problem?

DUNAWAY: The idea of a licensing agreement is the central focus of the issue. I think there are some problems with the use of licensing agreements in the mass market.

I'll give you a little background on the problem. License agreements are essentially saying that you are being given the right to use a particular application, you do not own it nor do you have any exclusive rights to the product.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

You pay a price and as long as you perform in accordance with the terms of the license agreement you are allowed to use it. The customer may only copy for backup purposes, etc.

This is all well and good in the situation of the buyer and seller sitting down and negotiating the agreement. They reach an understanding and the buyer should be held responsible. This not the case in the mass market.

PJ: You mean it is just assumed that the user agrees to the licensing terms?

DUNAWAY: Right. In the mass market situation it is just a take it or leave it proposition. There is no bargaining, no 'meeting of the minds'' , and no negotiating of the purchase. This comes awfully close to what could be called an unconscienable contract, or adhesion contract.

PJ: I remember the mention of a licensing agreement that was hidden in a shrink wrapped package, completely invisible to the buyer.

DUNAWAY: Any disclaimer, like in the electric appliance product business, must be very visible to be valid.

PJ: What kinds of cases does you office usually handle?

DUNAWAY: Our office is running into lots of program ownership problems, simply because people are not using 'work for hire'' agreements or partnership agreements. Something should be put down on paper to establish each partys' rights in regards to a piece of software.

PJ: Who seems to be complaining the most, the programmer or the publisher?

DUNAWAY: What we have found is that the publisher, who provides money and/or equipment to a developer, will end up with a dissatisfied programmer who goes elsewhere. The publisher has no source code to show for their investment. This is just a normal, everyday contract dispute. Verbal agreements in these areas are just the worst thing that either party can do.

PJ: What should a programmer do if he feels that a firm is selling 500 copies of his program a month but his royalty checks are only amounting to $90 every quarter?

DUNAWAY: Well, short of filing suit for an accounting, the proper approach is never to sign a contract if you do not understand the factors involved in the first place. Someone should review the contract whether it be a CPA or attorney. It is crucial if the programmer wants to get a decent deal.

PJ: If the programmer lives in Walrus, North Dakota then his chances of finding someone versed in software contracts are quite limited.

DUNAWAY: Certainly that is a problem. How is a programmer going to know about an accountability clause if he doesn't talk to an attorney? And even then, an attorney who is not versed in software law may not be aware of them, either.

PJ: Let's say the programmer calls you on the phone and wants an experienced quick review of a contract that they have been asked to sign. Does your firm do that?

DUNAWAY: We do this all the time. We work through the mail with a number of clients. What we do is just check the contract out. We know what to look for, what the programmer should request in a contract. This would normally not take more than a half an hour. Then we would write up a checkpoint list of suggestions and points to be negotiated. This will give him an idea of where he should end up in the contract.

PJ: What is a ball park price range for exactly this short review process?

DUNAWAY: There are not a lot of firms that specialize in software law. Their fees generally run in the $75 to $160 range. The higher prices are in the larger firms.

Our particular firm bills out at $90 per hour. We would probably review a contract, put together some correspondence, and this would take around 45 minutes to an hour. A simple review of a contract would run you around $70 to $120.

PJ: That seems quite reasonable considering a programmer is about to sign over a major investment of his time and creative energy.

DUNAWAY: It is incredible what people have brought to us. We have seen contracts that give away complete ownership, copyright and future ownership of the software. Besides that, they may be getting a raw deal on the royalty agreement as well. There are some terrible form contracts that are out there. It is a hard education to learn everything after the fact.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ: An explanation of what each side should reasonably ask for would help all of us. Would it be possible to write up a check list of this sort?

DUNAWAY: Maybe we could put together one.

PJ: Is there a way out once you discover that you signed away everything?

DUNAWAY: What we shoot for in a contract which avoids this problem is having the programmer ask for minimal sales guaranties, commitment of advertising funds and so on. This gives the publisher very strong contractual incentives to live up to his end of the bargain.

PJ: This puts pressure on the publisher to actively promote the product or to try and get out of the contract by his own choice.

DUNAWAY: And on the other hand, even if the publisher does not choose to get out, the programmer would still have the contractual right to withdraw. This is what we would like to see happen in a contract a programmer makes.

PJ: We all hear war stories about software writers losing their shirts because they failed to understand the rights they were signing away.

One other brief example. If a husband and wife software writer/publisher firm goes into business in California, they may discover a competitor in Colorado that has copied their software, changed the name and sold it as their own. What is the process for stopping this sort of infringement? It always tends to be out of state.

DUNAWAY: Typically it doesn't matter whether it is in or out of state. You will end up in federal court in most cases. The copyright laws are federal laws. If you had the minimal forsight to copyright your software then you don't have any problem about filing in Federal Court and making them come and defend.

PJ: They have to come and defend in your area?

DUNAWAY: That is a question of litigation venue. The Federal Courts are set up into regional districts which includes a number of states in your area.

PJ: Is there any single point that you feel would help everyone involved in software development and publishing?

DUNAWAY: Putting any business relationship into writing. Anything having to do with program development or work for hire. Don't do it verbally. Just writing something down forces you to think ahead and look at the future possibilities you must face. It is worth its weight in gold.

REMER, REMER & DUNAWAY also offer a book that is very valuable for clarifying the legal issues surrounding software rights. The focus is from the software writers standpoint. It is called "LEGAL CARE FOR YOUR SOFTWARE" by Daniel Remer and is available from:

REMER, REMER, & DUNAWAY
444 Castro St., Suite 400
Mountain View, CA 94041
ph. 415-964-7622

□

OPT-TECH SORT™
FORMERLY OPTIMUM SORT

SORT/MERGE program for the IBM-PC

— NEW Version 1.1 —
Contains all the most desirable features!

Features

- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Written in assembly language for **high performance**
- Supports variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- Can be run from keyboard or as a batch command
- Can be called as a BASIC subroutine
- Easy to use, includes on-line help feature
- Full documentation — sized like your PC manuals
- **$75** — VISA, M/C, Check, Money Order, COD, or PO
  Quantity discounts available

To order or to receive additional information write or call:

OPT-TECH DATA PROCESSING
P.O. Box 2167 Humble, Texas 77347
(713) 454-7428

Requires PC-DOS, 48K and One Disk Drive

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IMPRESSIONS OF DOS 2.0

by Morton F. Kaplon

I recently acquired the new Operating System, DOS 2.0 and thought it might be useful to pass along my impressions, comments, what have you.

Overall, I find it a significant improvement over 1.1. It is clear that the progression is towards a UNIX like system as reflected in the REDIRECTION and PIPING aspects included in the system, as well as the HIERARCHIAL DIRECTORY system which is really a boon and allows one to organize him/herself much more logically than previously possible. The inclusion of the SORT and FIND utilities as well as the PRINT SPOOLER also add significantly to system capabilities. Additional goodies are the MORE command which types just one screenful and prompts you to move ahead, also a variety of LOGICAL CHOICE commands usable in batch files which greatly enhance their capability. There are other operating commands that are useful, but the above are the most significant of the DIRECTLY IMPLEMENTABLE COMMANDS.I should also mention that if you do use different directories to organize your files, then I have found it most useful to use the PROMP utility (which allows you to change the system prompt) so that it automatically indicates which directory you are in.

There are in addition two significant advances, referred to as EXTENDED SCREEN AND KEYBOARD CONTROL (Chapt. 13 of the manual) and INSTALLABLE DEVICE DRIVERS (Chapt. 14 of the manual). These are both interesting and puzzling in many aspects.

First the big puzzlement. In chapter 14, they give as an example of how to write an Installable device driver, a 180K memory disk. I have copied the Assembly language program, assembled it and it runs marvellously. It is trivially modifiable to create a 360K memory disk, which I have also done and I find them both very useful. The puzzlement is, why did they not include them with their utilities. The .com program is small and if they had been included, then they would have had the excellent feature of including in the operating system, both a print spooler and a memory disk. Instead, they require you to take the time to copy and assemble the program. Incidentally, I am including on this disk the source code for each memory disk as well as the assembled code. The way it is written, the memory disk corresponds to drive D: on a system with two disk drives (C: appears to be reserved for the hard disk). There is an additional interesting aspect of the program, which I do not understand, and which caused me some trouble in getting it to run.

On page 14-28, line 108, is the code:

```
DB      'IBM  2.0'        ;Vendor Identification
```

When I transcribed the code, I (perhaps inadvertantly) left only one space between IBM and 2.0, i.e.:

```
DB      'IBM 2.0'
```

The program assembled OK, and when I called D: it went there and in fact the appropriate amount of memory had been reserved for the drive as verified by using CHKDSK A:. However, I could do nothing with drive D: of any utility and in fact trying to do something usually caused the system to hang up. The only difference I could find between my transcription and their text was that single space referred to above. I corrected it and everything has been running perfectly ever since. However, I am really puzzled as to why that one change should have the effect it did. Increasing the space between the M and 2 to a value larger than 2 did not adversely affect the running of the program. HELP!! on this one please.

I would not call chapter 14 a model of clear writing, but chapter 13 is something else again. The introduction to chapter 13 said, and I quote- "With DOS Version 2.00 you can issue special character sequences from within your program that can be used to control screen cursor positioning. You can also reassign the meaning of any key in the keyboard." I thought, that is great. Now I don't have to get PROKEY (or equivalent) to get Macro capability from the KB. Then I read the Chapter. If anyone can figure out how to implement the command structure just from the directions given, he is a genius indeed. It took me an enormous amount of nagging at IBM to get any clarification on this Chapter (they kept insisting it was crystal clear) and even after I got some information, it was incomplete, but enough to lead me to arrive at a method of implementation. What one has to do, at least for the Mode of Operation and KB reassignment, is as follows.

You create a .bat file which has in it the sequences given under the various headings on the chapter; include the command (as stated in the manual) DEVICE = ANSI.SYS in the CONFIG.SYS file and then, if the system was booted with the CONFIG.SYS file set up, either enter the command(s) for the given file(s) you have created or else include them in the autoexec.bat file for boot. The

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

critical thing is how to get the ESC character entered. The example given at the bottom of the page 13-2 is cryptic to say the least, but after having learned how to implement the commands otherwise, I finally understand what it means and implies. In addition to the method IMPLIED by that, there are at least two other ways to accomplish it.

1.VEDIT

a) Enter ESC from Visual Editor by using Next CHR Literal <ESC>

b) Follow this with the sequences outlined in the Chapter with the exception, that (see page 13-10) DO NOT ENTER ";" between the last control entry and p whether the last control entry is a "#" or a "string" .

c) The file must be named xyz.bat and if you want it automatically called, put it in the autoexec.bat file.

N.B. There is a peculiar problem with VEDIT in DOS 2.0. It only works from the root directory and with files in the root directory. In addition, it will not create a file of less than 128 bytes (which is a real problem as will be seen later). You can fix up the file size problem by running the file through EDLIN.

1.1 VEDIT also

A file labelled ESCAPE was created with VEDIT in which just <ESC> was included. This file was then edited with EDLIN to reduce the size and placed in the directory under the name ESCAPE. One just copies the file ESCAPE to a new name.bat and edits in (using EDLIN) the other control sequences desired. In essence, VEDIT was used just to created a seed file for <ESC>.

2. BASIC

One uses BASIC to either create the seed file as above, or a desired file with the appropriate sequences. The program is

```
OPEN "filename.bat" FOR APPEND AS #1
'Opens "filename.bat" for seq.input
PRINT #1,CHR$(27);"[ rest of commands p"
' Input command string ESC[ etc.
```

If there is a string inside the "[p" , then just enclose that string in single quotes, e.g. chr$(27); '[. . .'string'. . .p"

3. USING DEBUG (Method implied by example at bottom of 13-2.)

a) Create a file with the desired name, say xyz.bat from the console or from EDLIN and enter at least as many bytes (with blanks) as you will need for the commands you will use.

b) Call that file from DEBUG; i.e. DEBUG xyz.bat

c) Enter the desired series of commands from DEBUG using the enter command. I used the address they suggested and it works. That is e 200 1b "[p" for KB commands for example. any strings within the "..." should be contained in single quotes ' '

d) Write those commands back to the file you entered with by the command w ds:200 ; you have to use ds:200 since the e(enter) command enters the offset from ds, but the w(rite) command assumes cs.

e) Then enter the created file and you have the implementation. One could also of course use DEBUG just to create a seed ESC file as above.

After having finally solved the method of implementation I sat down one day to write a series of Macros that I thought would be useful. I wrote them up and checked each out individually and then put them together in the autoexec.bat file for booting. When I loaded the system it hung up, and I mean it really hung up. No CtrlAltDel would reset. It had to be turned off. Some experimenting indicated that the problem seemed to be that some of the resident system was being overlaid by the KB reassignment commands. So I did some experimenting and found that if the total file size of KB reassignments was kept below about 300 bytes, there was no problem, but that if it exceeded that the system would hang up as indicated. Again I checked back with IBM and they replied " Yep, that is true. If you enter more than 300 bytes in KB reassigments, the system is overlaid; that is all we could fit in." Well it was comforting to know I wasn't a total fool, but don't you

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

think they could have at least told you that in the manual. Oh yes, they still insist the manual is perfectly clear.

The final question is, how useful are they. Well they don't seem to work from anywhere but from DOS. If you set up a series of KB reassignments, then they work from DOS but not from BASIC, nor are they implemented in VEDIT or in Easy Writer II. Thus, they certainly do not have the transportability as such systems as PROKEY and others do.

All the other commands in chapter 13 can be implemented as above, though it makes little sense for the Cursor commands, since their use is limited from DOS and the instructions with respect to them in the Chapter via DOS function calls do not make much sense as stated.If you have to use DOS function calls, then you need an Assembler, and if you have that,you can accomplish these functions in other ways just as readily. It seems to me that these extended controls are really only meaningful if one can implement them without going through Assembly language. At the least they should be implementable from DOS and at the worst, via BASIC. Anyhow, there went my Macros. I have ordered PROKEY.

The last time I wrote you I indicated some puzzlement over the logic of certain statements in the DOS manual with respect to using INT 27. I have now found out how to implement a program and have it stay resident. One has the fixed address program called by the loaded program by entering it in the Program Prefix at location 05CH (p.E-6 of DOS 1.1 and B-10 also of DOS 1.1). This is included in the source file labelled TFR24C70.ASM which is the source file for a program TFR24C70.COM which prints on the screen at row 24,col 70 the symbol @ if CapsLock is toggled on, and at row 24,col 71 the symbol # if NumLock has been toggled on. Clearly the coordinates as well as the symbols and screen format are changeable. They are not as useful as one would hope from DOS because line 24 is scrolled. However from BASIC, where line 24 is not scrolled it is most useful, as well as from VEDIT and also in Easy Writer II (where I have put the flags on line 01) where the top two lines are not scrolled.

Before I close I have one or two final observations. The disk that I am using here is a single sided double density, which I have formatted as single sided, 8 track. However, if I put this in my machine (which has two double sided drives) and format it without the 1 side command, it formats up to the total capacity of some 320K odd bytes and works perfectly OK. I also found out the following, inadvertantly. I formatted a disk identified as single side, single density, and it worked to the 320K odd capacity. "It is a puzzlement" to quote the King of Siam.

Well it has been nice talking with you again. I find the magazine most informative and useful. I particularly enjoyed the articles on the Text Editors; I have VEDIT, as you note, and find it absolutely first rate. In fact I use it more than Easy Writer II, the only exception being when I am doing some work that requires sub or superscripts then the ease of entering those commands from EWII for the printer makes it preferable to VEDIT,though once I have PROKEY (assuming it works with VEDIT) I may use that to ease up the printer initializaiton for that aspect. Incidentally, I find the ability to split a line with CR highly desirable in editing and wish in fact that EW II had it. It makes many things much simpler. I am still looking forward to the issue on C.

P.S. If it does not violate any IBM caveats, I would be quite willing to supply copies of the runtime code for the DOS 2.0 memory disk to those wanting it if they send $5.00 to cover costs, or a disk to be copied and a stamped return mailer, as well as the flag program with the commands displayed on lines as requested.It should be noted that the memory disk programs only run under DOS 2.0, since they utilitze the Installable Device Driver aspects.

Morton F. Kaplon
11 White Birch Dr.
Pomona, NY 10970

C SCREEN EDITOR

CSE: A full-screen text editor written in C

$60 Includes the complete source code

- Powerful command set includes cursor control, find/replace, block move, file inclusion, and nested macro commands.
- Full-screen editing, including cursor control, insert/overtype, single and multiple line insertion and deletion, and forward and reverse scrolling.
- Commands to perform large-scale operations on the text buffer, such as find, find and replace, move the text pointer, read input, and write output.
- A macro facility which allows grouping of a set of commands for repeated execution.
- Ability to work on files larger than the text buffer size.
- Complete configurability of control sequences, such as cursor keys and editor control strings, to allow customization of CSE for a specific terminal. CSE is configured to use special editing and function keys on the IBM PC.

NMD Northwest Microsystem Design

P.O. Box 10853 • Eugene, OR 97401 • (503) 484-7129

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BEEP

BEEP & BEEPINT

by Art Zachai

These two routines illustrate several concepts in programming the PC in assembly language, and making use of some of it's resources. They are not written with the idea of being particularly useful programs, but rather, as a starting point for your own programs. The concepts illustrated are accessing the speaker, stay-resident routines, and some conditional assembly directives.

The BEEP routine can be assembled as a DOS .EXE file, and running it will cause the speaker to beep. It is similar to the beep routine in the BIOS, but with more comments and a different tone. It should be assembled with the conditional flag DOS_ENTRY = 1.

The BEEPINT routine is a stay-resident program. It uses some of the code in BEEP, and should be assembled with the BEEP conditional flag DOS_ENTRY = 0. The conditionals cause only the procedure code to be assembled in BEEPINT. Do not assemble BEEP with DOS_ENTRY = 0, it won't work. After assembling and linking BEEPINT, convert it to a .COM file using the EXE2BIN utility. After running the program, other programs can sound the beep by executing an INT 40H.

Notice that running BEEPINT more than once will cause multiple copies of the routine to be left in memory, with only the most recent copy having the INT 40H entry point. See if you can re-write BEEPINT so that it will first look through memory to see if it is already installed. If it is, exit normally, otherwise exit stay-resident. □

Three Ways to Learn
Assembly Language Programming
on the IBM-PC
from
ARTHUR ZACHAI

1. If you are proficient in assembly language on another machine, but need to *spin up* on the IBM-PC, attend the one-day **Assembly Language Workshop.** (Eight hours of intensive instruction and hands-on experience. $60.00)

Groups may take advantage of the workshop by providing students (12 maximum), space, and machines (one per 3 or 4 students), and paying a negotiable fee.

2. If you are proficient in a high level language, but new to assembly language, attend the six week **Assembly Language Programming** course. (Six evenings for a total of 18 class hours, with homework. $165.00)

3. If you prefer to learn by yourself, refer to **Twenty Sample Assembly Language Programs.** Useful and instructive routines written for the IBM-PC assembler and DOS, profusely commented and page number references to the IBM manuals. ($25.00)

All three methods will help you to understand:
- Assembler pseudo-op use
- Interrupts and DOS/BIOS structure
- Register use conventions
- Writing "readable" code
- Use of LINK and DEBUG (not in SAMPLE PROGRAM package)

The instructor has over 17 years of professional experience in engineering and computer science, two years with the 8086/88, and 10 years teaching experience at a local university.

Send name, address and phone number for course schedule to:
Arthur Zachai
3817 Veazey Street, N.W.
Washington, D.C. 20016

Now-chain Pascal programs and pass data between them with...

CHAIN™

Developed exclusively to improve Pascal programming efficiency on the IBM Personal Computer! This time-saving software package chains programs together to achieve the effect of increased RAM, *without* increasing your hardware! CHAIN saves you time and helps you develop more comprehensive menus. It's an absolute necessity for developing or expanding user-friendly programs.

10-Day Money-Back Guarantee!
See what CHAIN can do in your own programming applications. If not completely satisfied, return the diskette and manual within 10 days of receipt for a full 100% refund!

Order Today! Mail check or money order for $46.95, or call 1 (312) 298-3610

KEYSoftware Inc.

VISA MasterCard

Dept. 24 • 2350 E. Devon Ave. • Suite 138 • DesPlaines, IL 60018
Dealer Inquiries Invited

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
 The IBM Personal Computer MACRO Assembler 06-13-83     PAGE    1-1

BEEP ROUTINE

        SUBTTL BEEP ROUTINE
        PAGE 64,132
COMMENT*
        A ROUTINE TO SOUND THE BEEP

        THIS ROUTINE IS NOT MUCH DIFFERENT THAN THE ONE IN THE BIOS ROM,
        (TECH. REF, PG A-18) EXCEPT THAT THERE ARE MORE COMMENTS. WE CANT
        USE THE BIOS BEEP ROUTINE DIRECTLY, AS IT WAS ASSEMBLED AS A NEAR
        PROCEEDURE. WE COULD GET TO IT WITH A FAR CALL, BUT THE RETURN IS WRONG.
        THIS ROUTINE IS ALSO NEAR, BUT IT WILL BE WITHIN REACH OF 64K PROGRAMS.

        TO PROGRAM THE SYSTEM CHIPS DIRECTLY, YOU WILL PROBABLY WANT THE DATA
        SHEETS FOR THEM. THEY ARE IN "THE iAPX 86,88 USER'S MANUAL",INTEL CORP.
        YOU MUST ALSO CHECK THE SCHEMATICS AND TECH REF PAGES NOTED BELOW TO
        BE SURE YOU DON'T MESS UP SOME CONDITION DOS EXPECTS. HINT; STAY AWAY
        FROM TIMER 0 AND DMA CHANNEL 0, AS THESE DO THE MEMORY REFRESH.

        HARDWARE SET-UP:        PPI PORT B BIT0= TIMER 2 GATE
                                PPI PORT B BIT1= SPEAKER DATA
                                SPEAKER = SPKR_DATA- AND TIMER_2-

        THIS ROUTINE MAY BE USED AS A DOS .EXE PROGRAM TO SEE IT WORK, OR AS
        CODE FOR A PROCEDURE USED IN ANOTHER PROGRAM (SEE BEEPINT). THE
        CONDITIONAL ASSEMBLY OF THE DOS ENTRY INSTRUCTIONS IS CONTROLLED BY
        THE VALUE OF DOS_ENTRY:

DOS_ENTRY=0     ;SET THIS TO DOS_ENTRY=0 TO SKIP THE DOS ENTRY STUFF
                ;AT ASSEMBLY TIME.
.LFCOND         ; AND PUT BOTH CONDITIONALS IN THE LIST OUTPUT (WHICH YOU
                ; ARE READING) SO YOU CAN SEE WHAT I DID.
IF1
IF DOS_ENTRY                    ;PRINT OUT WHICH VERSION IS BEING ASSEMBLED.
%OUT BEEP, DOS ENTRY VERSION
ELSE
%OUT BEEP, INCLUDE VERSION
ENDIF
ENDIF
;.............................. EQUATES ..................................*

TIMER_PORT      EQU     40H             ;SEE TECH.REF. PG 2-23
TIMER_MODE      EQU     TIMER_PORT+3
TIMER_CTRL_BYTE EQU     10110100B       ;SELECT COUNTER #2,LSB THEN MSB LOAD,
                                        ;MODE 3,BINARY, SEE INTEL 8253 DATA SHEET.
TIMER_VALUE     EQU     2711            ;CLK INPUT TO CTR=1,193,000,
                                        ;DIVIDED BY 2711 = ABOUT 440 HZ

PPI_PORT        EQU     60H
PPI_CONTROL     EQU     PPI_PORT+3
PPI_PORT_B      EQU     PPI_PORT+1
```



```
                                        PAGE    1-2
BEEP ROUTINE

PAGE
IF DOS_ENTRY    ;SKIP EVERYTHING EXCEPT THE CODE IF NOT A DOS_ENTRY

;.............................. STACK ....................................

STACK   SEGMENT STACK 'STACK'
        DW 4 DUP(?)
STACK   ENDS
;.........................................................................
                ASSUME SS:STACK
ENDIF
BEEP_CODE       SEGMENT 'CODE'
                ASSUME  CS:BEEP_CODE

IF DOS_ENTRY                    ;CONDITIONAL ASSEMBLY OF DOS ENTRY SET-UP.
  BEEP  PROC FAR
        PUSH DS                 ;MAKE THIS A DOS .EXE PROGRAM WHICH BEEPS ONCE.
        XOR AX,AX               ;IF THE CONDITIONAL IS TRUE.
        PUSH AX

ELSE
  BEEP  PROC    NEAR    ;OTHERWISE, JUST ASSEMBLE THE PROCEDURE.
ENDIF
        ASSUME CS:BEEP_CODE

        MOV     AL,TIMER_CTRL_BYTE  ;SET UP TIMER #2 TO DO A SQUARE WAVE.
        OUT     TIMER_MODE,AL
        MOV     AX,TIMER_VALUE
        OUT     TIMER_PORT+2,AL     ;LOAD LSB
        MOV     AL,AH               ;CANT OUTPUT BYTE FROM AH.
        OUT     TIMER_PORT+2,AL     ;LOAD MSB. TIMER IS NOT RUNNING YET.

        IN      AL,PPI_PORT_B       ;GET CURRENT VAULE OF PORT B, SET BY DOS OR BIOS.
        MOV     AH,AL               ;SAVE IT TO RESTORE LATER. THE PPI CHIP SHOULD
                                ;BE SET UP FOR MODE 0, SEE INTEL 8255 DATA SHEET
                                ;AND TECH. REF. PG.2-24.
        OR      AL,03           ;TURN ON TIMER GATE AND SPEAKER DATA BIT, GATE STARTS
        OUT     PPI_PORT_B,AL   ;TIMER, TIMER OUPUT IS ANDED WITH DATA BIT=1, MAKING THE
                                ;SQUARE WAVE COME OUT THE SPEAKER.

        XOR     CX,CX           ;CLEAR CX FOR USE AS LOOP COUNT,
DLY:    LOOP    DLY             ;CAUSES LONG LOOP OF DO NOTHING.

        MOV     AL,AH           ;GET THE ORIGINAL CONDITION OF PORT B BACK,
        OUT     PPI_PORT_B,AL   ;AND RESTORE IT. CANT RESTORE TIMER, SINCE THE MODE
                                ; REGISTER  CANNOT BE READ.
        RET
BEEP    ENDP

  BEEP_CODE ENDS
            END
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
 The IBM Personal Computer MACRO Assembler 06-16-83     PAGE    1-1
INSTALL A DOS STAY RESIDENT ROUTINE TO "BEEP" ON INT 40H


TITLE INSTALL A DOS STAY RESIDENT ROUTINE TO "BEEP" ON INT 40H
PAGE 64,132
COMMENT*

   THIS ROUTINE INSTALLS THE BEEP AS AN INTERRUPT ROUTINE, AND STAYS RESIDENT.

PROGRAMMER:     ZACHAI
DATE WRITTEN:   1-20-83
CHANGE HISTORY:


 AT INSTALLATION, THE BEEP ADDRESS ENTRY IS USED TO SET THE INTERRUPT 40H
VECTOR. THE PORTION OF THE PROGRAM TO STAY RESIDENT IS MOVED DOWN IN MEMORY,
INTO WHAT WAS THE PROGRAM PREFIX SEGMENT. THE ROUTINE THEN EXITS/STAY RESIDENT,
SAVING THE BEEP CODE, BUT NOT THE INSTALLATION CODE.

THE REASON FOR MOVING IT DOWN IS MERELY TO REDUCE THE AMOUNT OF MEMORY TAKEN
OVER BY THE ROUTINE.  IT IS MOVED DOWN TO 5CH SINCE THIS ROUTINE WILL NOT TRY
TO DO ANY DISK I/O, HENCE THAT SPACE WOULD BE WASTED.  IT IS NOT MOVED ALL THE
WAY TO ZERO, SINCE THERE IS A "RESERVED" AREA IN THERE.  IS ONE OF THOSE
MYSTERIOUS 'RESERVED' AREAS IN THERE.  (SEE DOS 1.1 PG.  E-6)

CALLING CONVENTION:

        INT 40H ;AX AND CX WILL BE DESTROYED.

NOTES:

THE PROGRAM IS ASSEMBLED AND LINKED AS USUAL (IGNORE THE NO STACK MESSAGE),
AND THEN CONVERTED TO A .COM FILE USING EXE2BIN.

                                MEMORY LOOKS LIKE THIS:

            -AFTER BEEP_INT IS LOADED                   -AFTER INT 27H

            ┌──────────────────────────┐                ┌──────────────────────┐
            │  RESIDENT COMMAND.COM    │                │ RESIDENT COMMAND.COM │
 XXXX:0000  ├──────────────────────────┤                ├──────────────────────┤
            │  BEEP_INT PROG.PREFIX    │                │                      │
            ├──────────────────────────┤ ;005C          │  BEEP_INTR:          │
 XXXX:0100  │  JMP INSTALL_BEEP        │                │     CALL BEEP        │
            │  BEEP_INTR.:             │                │     IRET             │
            │       CALL BEEP          │                │  BEEP PROC           │
            │       IRET               │                │       :              │
            │  BEEP PROC               │                │     ENDP             │
            │       :                  │                ├──────────────────────┤
            │     ENDP                 │                │                      │
            │  INSTALL_BEEP:           │                │                      │
            │       :                  │                │                      │
            │     INT 27H              │                │                      │
            └──────────────────────────┘                └──────────────────────┘
*
```

```
 The IBM Personal Computer MACRO Assembler 06-16-83     PAGE    1-2
INSTALL A DOS STAY RESIDENT ROUTINE TO "BEEP" ON INT 40H


PAGE

BEEP_CODE       SEGMENT 'CODE'
                ASSUME  CS:BEEP_CODE

                ORG 100H            ;SEE EXE2BIN IN DOS MANUAL.
INSTALL_BEEP_ENTRY:
                JMP INSTALL_BEEP ;WE WANT BEEP TO LOAD ABOVE THE INSTALL
                                 ;ROUTINE, AS THE INSTALLATION PART IS NOT
                                 ;GOING TO STAY RESIDENT.

;..............  COME HERE ON INT 40H, AFTER INSTALLATION  ...................

BEEP_INTERRUPT  LABEL   NEAR
                CALL    BEEP
                IRET                ;AND RETURN.
;-------------------------------------------------------------------------

;LIST OFF, INCLUDE BEEP.ASM, LIST ON.



 The IBM Personal Computer MACRO Assembler 06-16-83     PAGE    1-3
INSTALL A DOS STAY RESIDENT ROUTINE TO "BEEP" ON INT 40H
BEEP ROUTINE

.LIST                   ;TURN THE LIST BACK ON
;       ┌─────────────────────────────────────────────────────┐
;       │                                                     │
;       │     THE ASSEMBLER PUTS THE ACTUAL BEEP ROUTINE HERE │
;       │                                                     │
;       └─────────────────────────────────────────────────────┘
;-------------------------------------------------------------------------
INSTALL_BEEP    LABEL   NEAR

;       MOVE THE CODE THAT WILL REMAIN RESIDENT DOWN TO 05CH

CODE_LENGTH  EQU (INSTALL_BEEP-BEEP_INTERRUPT+1)/2 ;# OF WORDS TO MOVE.

        MOV     CX,CODE_LENGTH
        MOV     DI,05CH     ;DESTINATION INDEX = AREA IN PROG PREFIX.
        MOV     SI,OFFSET BEEP_INTERRUPT
        REP     MOVSW       ;ES IS POINTING TO TOP OF PROG PREFIX (.COM FILE).

;       SET VECTOR AT INT 40H TO POINT TO BEEP_INTERRUPT. SINCE THIS MODULE
;       WILL BE A .COM PROGRAM, DS AND CS ARE THE SAME,AND THE OFFSET IS 05CH.

        MOV     DX,05CH
        MOV     AX,2540H    ;SET ADDRESS IN DS:DX AS VECTOR FOR INT 40H
        INT     21H

        MOV     DX,DI       ;SET DX TO LAST INSTRUCTION OF PROGRAM+1, IN DI FROM
        INT     27H         ;MOVSW. LEAVES THE BEEP STUFF, WIPES OUT THE INSTALL PART.
BEEP_CODE       ENDS
                END INSTALL_BEEP_ENTRY      ;SETS ENTRY POINT OF MODULE.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C INTERFACE

C86

The enclosed program may be of help to someone interfacing the CI-C86 C compiler to .asm routines. The examples show simple methods of accessing the joysticks and buttons. "C" is too slow for this kind of work, as it returns a very small range of values.

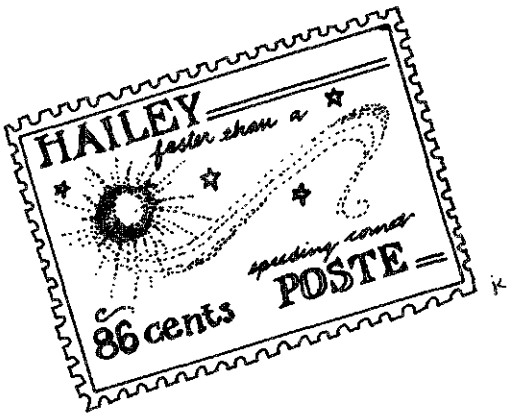
You should then use the CI utility to convert the .OBJ file to a .O file (CI internal format.)

Keep up the good work. Your magazine gets good reviews from the "pro's" on Compuserve.

(The best spot on the Compuserve Information Service is the IBM SIG . . . . of course. Used to be Computers and Electronics had their own group. There is also a Programmer's Group, but it has just been started and has a long way to go. To get to the IBM SIG on CIS just go to the Personal Computer Menu, pick out Special Interest Groups and choose IBM PC.)

Bill Hailey
Carenco, LA

```
CODE SEGMENT BYTE PUBLIC
        ASSUME CS:CODE

;=================================================================
;= C interface to Joystick                                       =
;= format :   joy(intnum);                                       =
;=              int num is mask 1=ax,2=ay,4=bx,8=by              =
;=                                        bill hailey   4/25/83 =
;=================================================================
        PUBLIC JOY      ; so C can find
JOY:
        PUSH BP
        MOV BP,SP       ;MAKES BP+4 PASSED VALUE
        MOV DX,201H     ;PORT PAST FF USES DX
        MOV AX,0        ;VALUE TO SEND OUT
        OUT DX,AL       ;SEND IT OUT
; time out
        MOV CX,00       ;INITIALIZE COUNTER
JOY1:   IN  AL,DX       ;TOP OF LOOP INPUT BYTE
        AND AX,4[BP]    ;MASK WITH BIT PASSED FROM ROUTINE
        JZ  FINIS       ;IF TIMED OUT LEAVE
        INC CX          ;ELSE INCREMENT COUNTER
        JMP JOY1        ;LOOP
FINIS:  MOV AX,CX       ;REPORT RESULTS
        POP BP
        RET

;=================================================================
;= C interface to Joystick  Buttons                              =
;= format :   button(intnum);                                    =
;=              intnum is mask 16=a1,32=a2,64=b1,128=b2          =
;=              returns 0 if true       bill hailey   4/25/83 =
;=================================================================

        PUBLIC BUTTON
BUTTON:
        PUSH BP
        MOV BP,SP
        MOV DX,201H     ;PORT PAST FF USES DX
        MOV AX,0        ;VALUE TO SEND OUT
        OUT DX,AL       ;SEND IT OUT
        IN  AL,DX       ;INPUT BYTE
        AND AX,4[BP]    ;MASK WITH BIT
        POP BP
        RET


CODE    ENDS
        END
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE PASCAL SIG

The Pascal program that follows is part of an excellent tutorial diskette available from the Capital PC Monitor users group in Washington, DC. To obtain a copy of the complete diskette, send $6.00 to :

Capital PC Software Exchange
P.O. Box 6128
Silver Springs, MD 20906
*(Be sure and ask for disk #*10, the Pascal Primer)

The tutorial disk provides a number of sample Pascal programs. Some are very simple; others are more complex. Most are quite short, while a few are are fairly long. All will compile on a 128K machine with two regular (160K) disk drives.

These programs were put together to help the programmer get started with IBM Pascal. Many aspects of the language are hard to figure out from the reference manual; having some sample programs that do work is frequently very helpful. These programs were assembled, therefore, mainly as illustrations of how various things can be done in IBM Pascal. Most are intended more as "helpful hints" than as products.

In rough order of increasing complexity the programs provided are:

COPYFILE -
A fairly basic Pascal program that shows how text files are defined, read, and copied.

DUMPFILE -
A more complex Pascal program, which will produce a combined hexadecimal and ASCII dump of a disk file.

TABSET1 -
A very basic Pascal program which shows how the printer can is accessed, and which sets tabs on an Epson MX-100.

TABSET2 -
Another version of the same, which treats the printer as a binary file rather than a text file.

TIMM, -
TIMU,
TIMM
A group of files, which illustrate the concept of a Pascal unit. File TIMU is the unit itself; file TIMI is the interface; and file TIMM is a main program which uses the unit. The unit itself illustrates use of the DOSXQQ function to call on DOS services.

SCREEN -
A short program to illustrate how the ADS type can be used to get at specific addresses, in this case the memory block associated with the monochrome display screen.

PRINTER -
A useful utility which can be used to print a whole series of files. (This program also uses (INCLUDEs) PARSE.P, INDEX.P, and DSNAME.P). A good example of a fairly long Pascal program.

XREF -
Another useful utility - a cross-reference generator. Also a longish Pascal program.

UNSQ -
A program for unsqueezing files, compressed by techniques such as Huffman encoding. Illustrates some fairly complex data structures.

PRETTY -
A Pascal pretty-printer. Useful utility for listing Pascal programs. Fairly long (Includes PARTA, PARTB, and PARTC) and fairly complex.

GETDIR -
A brief Pascal program that uses an assembly language routine (GETSEC), provided in both source and object, to show how one can access a disk directory through Pascal and Assembler.

VIDEO -
A short assembler subroutine which can be called by Pascal (as in the foregoing example) if you need to perform BIOS video interrupts from Pascal.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PASCLG, -
PASCL
PASC
A set of command files that help the whole process of compiling, linking, and executing Pascal programs. For use, these procedures should be copied on each of the three Pascal disks (PAS1, PAS2, and PASCAL library); your work disk should be in drive B, which also should be the default volume. A Pascal compilation, linking, and test of program TESTX.PAS can then be requested with the command "a:pascig testx"

FLUSH -

A short command file useful for cleaning up the residue of a Pascal compilation and test (.OBJ, .LST, .MAP, .EXE etc. files) A call to "flush xyz" will scratch all XYZ.—- files except for XYZ.PAS

That's it for now. If you find these programs and examples useful, the Pascal SIG will try to put together another batch. We hope that next time we'll include some of your contributions and suggestions!!!

Good Luck !

Pascal Special Interest Group
Capital PC Users' Group

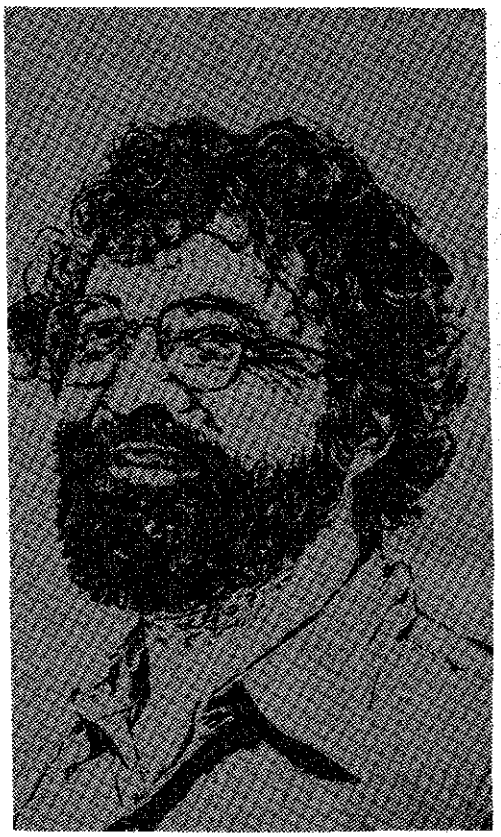
□

Disk Explorer
for your IBM/PC

- goes well below D.O.S.
- can read/write any sector
- is not stopped by copy-protect schemes
- recognizes D.O.S. Formats (1.0, 1.1, 2.0)
- recognizes 10-SECTOR JFORMAT

package includes 18-page manual

Available from computer stores or for $100 from:
Richard Larratt and Associates Ltd.
RR #1 Demorestville
Ontario, CANADA K0K1W0
Phone 613-476-5309



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
{getdir5 -- get disk directory sectors
 (C) Copyright 1982 by Walter H. Rauser
        getdir5.pas -- version using pascal convention for buffer.
                       under development  (whr  5-16-82)
                       just recovered from FAT crash caused by Linker!!!
                       exchanged disks as directed because file not on disk

}
PROGRAM TESTGETDIR(input,output);
type    drivetype = (A,B);
                dirtype = array[1..112] of record   {DOS 1.10 Double Sided Drives}
                                        name    [0]:string[8];
                                        ext     [8]:string[3];
                                        attr    [11]:byte;
                                        resvd [12]:array[1..10] of byte;
                                        time  [22]:array[1..2] of byte;
                                        date  [24]:array[1..2] of byte;
                                    cluster1 [26]:array[1..2] of byte;
                                        size1 [28]:array[1..2] of byte;
                                        sizeh [30]:array[1..2] of byte;
                                 end;

var             dir1 :dirtype;
                drive :drivetype;

{EXTERN declarations}
                function GETSEC( drive,first,numcf :integer;
                                 var buffer :dirtype;  var errorc :integer
                               ):boolean; EXTERN ;

{GETDIR - function to get directory sectors from disk.}
                function GETDIR( drive :drivetype;
                                                var dir :dirtype ):boolean;

                var  dr, errorc :integer;

                begin {getdir}
                        dr := ord(drive);
                        {add code to get FAT and test for double sided drives}
                        {only get first four file sectors to show how it's done}
                        if getsec( dr, 3, 4, dir, errorc ) then getdir := true
                        else begin
                                getdir := false;
                                writeln('Get Directory Sector Failed');
                                writeln('ERROR CODE = ', errorc:6:16 );
                             end;
                end; {getdir}

{-------------------------------------------------------------------------}

var     i,j :integer;
begin {testgetdir}
        for i:= 1 to 112 do dir1[i].name := '        ';
        drive := B;
        if getdir( drive, dir1 ) then begin
                for i := 1 to 64 do
                        with  dir1[i] do
                        {
                        write(i:5,'  ', name,'.',ext,'  ',attr:2:16,'  ');
                        for j:=1 to 10 do write(resvd[j]:2:16);     write('  ');
                        for j:= 1 to 2 do write(time[j]:2:16);      write('  ');
                        for j:= 1 to 2 do write(date[j]:2:16);      write('  ');
                        for j:= 1 to 2 do write(cluster1[j]:2:16);  write('  ');
                        for j:= 1 to 2 do write(size1[j]:2:16);     write('  ');
                        for j:= 1 to 2 do write(sizeh[j]:2:16);     write('  ');
                        writeln;
                        }
        end
        else abort('Aborting in TESTGETDIR',1,1);
end. {testgetdir}
```

```
PAGE 50,132
TITLE GETSECS   -PASCAL FUNCTION TO GET DISK SECTORS
;GETSEC --  Pascal Function to get sectors from disk using DOS's INT 25H.
;                       interfaces to IBM PC Pascal v1.00 Compiler (Microsoft).
;                       see DOS Manual, page D-25; and Tech Ref Manual, page A-32.
;********** (c) Copyright 1982 by Walter H. Rauser   5-16-82   **********
;{sample of Pascal declarations to use GETSEC}
;function GETSEC(drive  :integer;       {A is 0, B is 1, C is 2, D is 3}
;                                first  :integer;   {first sector # in 0-origin}
;                                numcf  :integer;   {number of sectors to get}
;                                var buffer :dirtype; {buffer to hold all fetched sectors}
;                                var errorc :integer  {DOS error code}
;                                                  ):boolean; EXTERN;
;{sample of buffer type for directory sectors, DOS 1.10}
;type dirtype = array[1..112] of record        {112 for DS Drives, 64 for SS}
;                                        name    [0]:string[8];
;                                        ext       [8]:string[3];
;                                        attr    [11]:byte;
;                                        resvd  [12]:array[1..10] of byte;
;                                        time    [22]:word;
;                                        date    [24]:word;
;                                  cluster1  [26]:word;
;                                  size1ow   [28]:word;
;                                  sizehigh  [30]:word;
;                                end;
; frame contents for call from Pascal
; offset len  variable     attributes
; [BP]+14  2  drive                int value parameter
; [BP]+12  2  firstsector          int value parameter
; [BP]+10  2  numofsectors         int value param
; [BP]+ 8  2  buffer               var param
; [BP]+ 6  2  errorcode            int var param
; [BP]+ 4  2  CS:                  return address long call
; [BP]+ 2  2  offset    return address
; [BP]+ 0  2  old frame pointer storage
;
;
MYSEG   SEGMENT PARA    PUBLIC  'CODE'
                ASSUME CS:MYSEG
                PUBLIC GETSEC
;
GETSEC  PROC    FAR
                PUSH    BP                      ;save old frame pointer
                MOV             BP,SP           ;set new frame pointer
                MOV             AX,[BP].14      ;put drive num into AL
                XOR             AH,AH
                MOV             CX,[BP].10      ;num of sectors to fetch
                MOV             DX,[BP]+12      ;logical record # of first sec, 0-origin
                PUSH    DS                      ;restore before ret
                PUSH    SS                      ;buffer is in stack segment
                POP             DS                ;DS: of buffer (transfer address)
;
                MOV             BX,[BP]+8       ;offset of buffer
                INT             25H               ;absolute disk read, DOS function.
                JC              ERROR           ;error if carry flag set
                MOV             AL,0[H          ; okay, set AL to return GETSEC=true
                JMP             DONE
;
ERROR:  MOV             BP,[BP].6               ;adr of errorc
                MOV     [BP],AX                 ;dos error code
                MOV             AL,0FFH           ; error, set AL to return GETSEC=false
;
DONE:   XOR             AH,AH
                POPF                            ;int 25H left Flags on stack
                POP             DS                ;restore DS
                POP             BP                ;restore frame pointer
                RET             10              ;clear parameters from stack
GETSEC  ENDP
MYSEG   ENDS
                END
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WHILE-WEND

WHILE / WEND

by Dan Rollins

I'm sending along a set of interlocking macros that implement a WHILE/WEND construct for assembly language programmers. You'll have to wait for my book (an 8088 Assembler book) to get the details of how it works, but in brief, it implements a "symbol stack" to enable communication between macros and allow nesting of WHILE/WEND loops. The main "trick" is the use of the ORG statement in the WEND macro. It "goes back" and replaces some NOPs generated by the WHILE macro so as to provide a loop exit when the condition become false.

The block of code between a WHILE macro and the nearest WEND macro will be repeated until a given condition is met. If the condition is false at the start, then the body won't be executed at all.

The condition codes are the letters of the conditional jump mnemonics sans the "J" , and the operands to compare can be various combinations of registers, memory variables or literal values, but literal values can only be specified as the second operand and the operands must not both be memory variables. Also, the operands must be of like type. In summary, just use the same rules you do for the CMP opcode.

Some examples:

```
while    ax, LE, 100          ; but not:  100 G ax
 while    word_var NE bx      ; (spaces are as good as commas)
  while    byte_var E  bl
   while    al ne (byte ptr es:[si])  ;() to keep an operand together
    call     collect_to_mama
    inc      si
   wend
  wend
 wend
 inc      ax
wend
```

A set of "structure" macros, including IF/ELSE/ENDIF and FOR/NEXT (a real gem) is fully described in my forthcoming book: "IBM-PC: MACRO Assembler Programming" to be published by Macmillan in 1984.

Here's a short macro that you might find yourself using in every program you write from now on. See if you can figure out what it does:

```
dc_if    macro cond,t1,t2,t3,t4,t5
         local tag
         jn&cond,tag
         t1
         t2
         t3
         t4
         t5
tag:
         endm
```

Hint: Remember that angle brackets < > can be used to keep several words of text as a single macro argument.

□

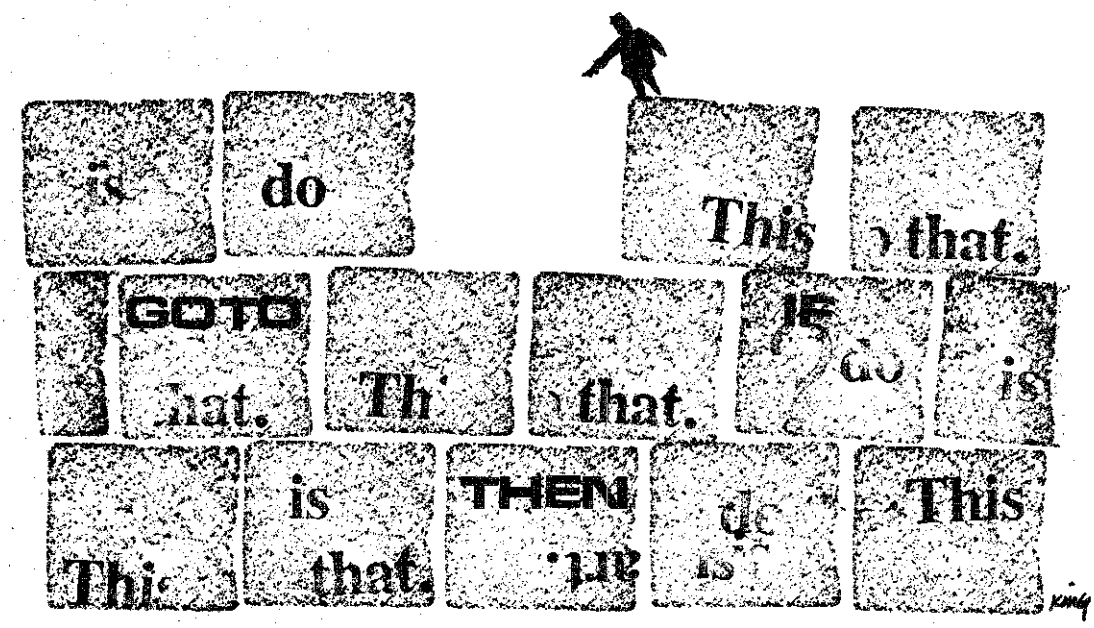
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

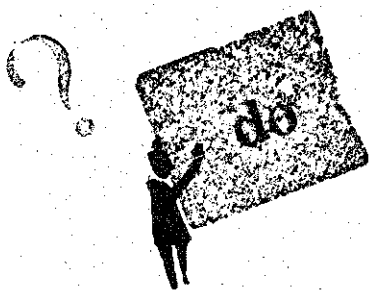
```
; Copyright 1983 by Dan Rollins

;---------------------------------
; WHILE (op1),(condition),(op2)
;
; With WEND, sets up a loop that repeats as long as the
;  comparison of (op1) and (op2) meets the (condition).
;
; (condition) is: E, NE, L, LE, B, BE, G, GE, A, AE (and all Nxx)
; (op1) is      : memory variable, register
; (op2) is      : register, immediate (,memory if op1 is register)
;
; Must resolve ambiguous memory references.  Example:
;      WHILE (byte ptr [SI]), L, 'Z'
; (() used here to keep (op1) as a single parameter)
;
; Requires all three parameters, alters only flags register
; Uses SAVE_THIS_ADDR and GET_LAST_ADDR macros

WHILE   macro   op1,cond,op2
        local   body

        SAVE_THIS_ADDR
        cmp     op1,op2
        J&cond  $ + 5
        SAVE_THIS_ADDR
        nop
        nop
        nop
        endm

;---------------------------------
; WEND  (no parameters)
; The continuation point after exit from a FOR or WHILE loop.
; Same action for WEND and NEXT (invoked either way)
; Preserves all registers

WEND    macro

        GET_LAST_ADDR
back_up =       last_addr

        GET_LAST_ADDR
        jmp     last_addr
exit    =       this near

        org     back_up
        jmp     exit

        org     exit
        endm

;-------------------------------
```

```
;-------------------------------

addr_num = 0  ;used by FOR/NEXT,WHILE/WEND,$IF/$ELSE/$ENDIF

addr_1 = 0    ;These define the "stack" used to pass data
addr_2 = 0    ; between macros.  Maximum nesting level is
addr_3 = 0    ; determined by the number of variables here.
addr_4 = 0    ; FOR/NEXT loops need two, WHILE/WEND and
addr_5 = 0    ; $IF/ELSE/$ENDIF need one per nest level.
addr_6 = 0    ;10 variables will support 5 levels of FOR/NEXT
addr_7 = 0    ; or 4 levels of FOR/NEXT and 2 $IF/$ENDIFs, etc.
addr_8 = 0
addr_9 = 0
addr_10= 0

;---------------------------------
; SAVE_THIS_ADDR   (no parameters)
; PUSH_ADDR %addr_num
;
; "Primitive" macros invoked by FOR, WHILE, $IF, and ELSDO.
; to temporarily store address values on a "symbol stack".
;
SAVE_THIS_ADDR macro
  addr_num = addr_num +1
  PUSH_ADDR %addr_num
endm
PUSH_ADDR macro num
  ifndef addr_&num
    .lall ;**** structure macro ERROR: nesting too deep ****
    .sall
    %out    **** structure macro ERROR: nesting too deep ****
  endif
  addr_&num = this near
endm
;---------------------------------
; GET_LAST_ADDR   (no parameters)
; POP_ADDR %addr_num
; "Primitive" macros retrieve addresses stored by SAVE_THIS_ADDR.
;
GET_LAST_ADDR  macro
  POP_ADDR  %addr_num
  addr_num = addr_num -1
endm
POP_ADDR macro num
  ifndef addr_&num
    .lall ;**** structure macro ERROR: too many NEXT,WEND,$ENDIF ****
    .sall
    %out    **** structure macro ERROR: too many NEXT,WEND,$ENDIF ****
  endif
  last_addr = addr_&num
endm

;---------------------------------
; DISPLAY_MACRO_ERR %addr_num
; Invoke at end of assembly to display if all nested
; structures were properly closed.
;
display_macro_err macro num
  %out macro nesting errors: num
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STRUCTURED PROGRAMMING





BASIC

SOME SUGGESTIONS FOR CODING IN IBM PC/BASIC

by Joe Raybawk Portland, OR

Let us start with a review (or introduction) of Structured Programming. Then, we will see how the various structures can be coded in PC/BASIC. I will try to throw in observations about which parts work well and which parts don't.

OK, so what's Structured Programming?

This is currently a $495-695 question. That is, you could pay something like that much to attend a one or more day seminar to really be inundated with it. Since you aren't paying that much let me give you the plain truth.

Structured Programming is a discipline of programming that uses a small number of the potentially huge set of possible ways of coding general control flow through a program.

The purpose behind Structured Programming is to make code clearer than it might be otherwise.

It is based on a theorem which proves that any computer program can be implemented using only three fundamental control structures. It turns out that, while three are all that is necessary, it is convenient to augment the basic set. You can implement the others using the basic three. The implementation is usually not as concise and often not as clear as just allowing the extras.

The goal here is not to follow the Theorem as a tenet of a programmer's religion but to create more reliable and more easily understood and modified programs.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Important Coding Structures

We will discuss the following coding structures:

The Fundamental Three:
SEQUENCE (also known as CONCATENATION),
IF-THEN-ELSE,
DO-WHILE,

Other Frequent Ones:
DO-UNTIL,
IF-THEN,
CASE,
GENERALIZED LOOP,
ERROR-CHECK LOOP.
and finally, the SUBROUTINE.

The general approach (or structure) of the discussion will be:
- Introduce the structure
- Explain its function, purpose or use
- Give examples of how to actually code it in BASIC.
- Comment as appropriate on how well BASIC implements it.

A KEY CONCEPT

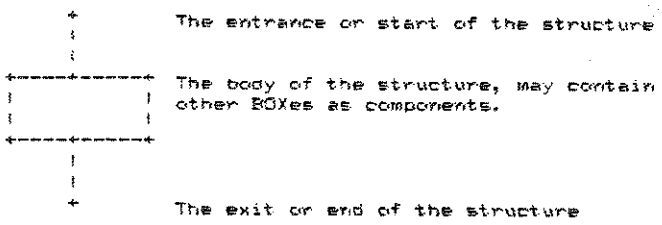
You will find that all of these structures have one important characteristic. For each structure, there is only one allowed way to get into the structure and one allowed way to get out of it.

This is key for two reasons. First, code that obeys this rule is much easier to follow. Second, it is easier to identify or state the conditions expected to be true if the code is actually executed. The behavior of a program is dependent on both its code and on the values of variables it uses. Hence, it can be very useful to know what the expected values of the variables are.

Aside: Structured Programming is often used to refer to more than what I am using it for here. Some people would prefer to call the aspect we are treating Structured Coding. If you are of such a bent, fine, call this whatever you like. You will find that these coding practices facilitate all of the other practices that are bundled into Structured Programming.

Let me introduce THE BOX. THE BOX is the global representation of the concept of One-Way-In and One-Way-Out. I will draw (draw?) this with the Way-In at the top and the Way-Out at the bottom. The reason for this convention is mostly because all the programming languages I know execute statements going down the page unless otherwise influenced. (Also, there is a pun about Top-Down Programming lurking here.)

A Picture of THE BOX


The entrance or start of the structure
The body of the structure, may contain other BOXes as components.
The exit or end of the structure

STRUCTURE: SEQUENCE or CONCATENATION

Introduction

This starts out being the simple concept of a set of statements which are executed one after another. In BASIC, this might be several lines of code with consecutive line numbers with no GOTOs or GOSUBs and no other code that branched into them.

The concept has to be enlarged a little to include a set of control structures that follow each other in sequence. Take it on faith for the moment that we will soon be describing a series of control structures that have only one-way-in and one-way-out. If we were to connect the way-out of the first one to the way-in of the second, we could describe the connected pair as having one-way-in and one-way-out. If the two structures were represented by BOXes, we could draw a bigger BOX around them. If we did not look inside this big BOX, we could view it as an allowed structure.

Purpose or Use

Sequence is kind of a fundamental way we have of building any logical structure. Also, recall that the purpose of all of this is to make it easier to understand what a piece of code is supposed to do. Let me relate back a little to the idea of knowing what the expected values are upon entrance to a section of code. If we know what the data is upon entrance to BOX-1 and we can figure out what BOX-1 does, we can determine what the data should be upon exit from BOX-1. This, of course, is the same value it has upon entrance to BOX-2. Now, if we can figure out what BOX-2 does, we can determine what the data should be upon exit from BOX-2. Hence, upon entrance to the BOX-1-BOX-2 combination with certain data values, we can determine

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the expected exit values. Since the BOX-1-BOX-2 combination has one entrance and one exit, we can treat it as one big BOX.

I have gone off the deep-end a little here in that I am plumbing the deep fundamental logic of why-how-what is a structure. If you have been able to follow this, you should be able to do the deep thinking about the rest of the structures on your own. (Also, I do not have to type as much.)

BASIC Coding Example

There are two ways to do this in PC/BASIC. The first is just to use a sequence of statements with ascending line numbers. The key is that there are not GOTOs (or THEN line-number) that take you out of the set and there are no GOTO or THEN line-numbers that take you to other than the first line in the sequence.

```
100 REM This is the start
110 A = 1 + B*3
120 PRINT "A = " ;A
130 REM This is the end
```

PC/BASIC also supports a form of Horizontal Concatenation using the colon to separate the statements.

```
500 A = 1 + B*3:PRINT "A: ";A:REM Everything after a
REM is a      comment: Including this.
```

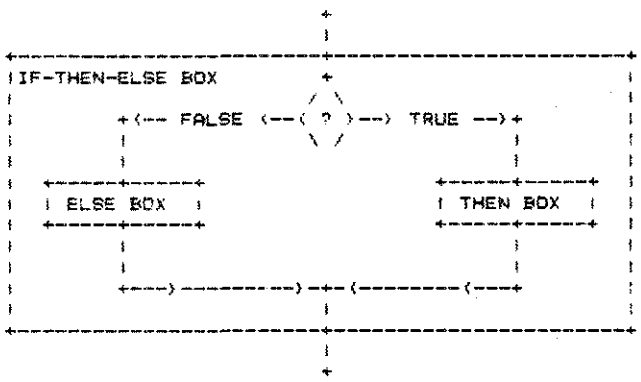
Comments

I included line 500 above to emphasize that the colon is only a partially effective line separator. In particular, the colon within the quotes after the PRINT is obviously not meant to function as a separator. It is not obvious that the colon after the REM should be treated as a literal. It is however. (Actually, the BASIC interpreter more-or-less ignores everything after a REM.)

STRUCTURE: IF-THEN-ELSE

This consists of the following: the entrance to the structure leads to code which evaluates some condition. If the condition is true, one BOX of code is entered. If the condition is false, the other BOX of code is entered. The end of each BOX must get to a common final point which is the end of the IF-THEN-ELSE structure.

IF-THEN-ELSE structure.



Purpose or Use

Allows us to do one of two sets of code based on the encountered data having certain values. Clearly, we determine by the decision or condition evaluation that the data has such characteristics. From this we determine which set of code should process it.

Example of BASIC Code

Simplest example: IF cond THEN statement-1 ELSE statement-2 where statement-1 and statement-2 are simple statements.

Please See the discussion below the IF-THEN STRUCTURE for hints about some problems with multiple statements in the boxes representing statement-1 and statement-2.

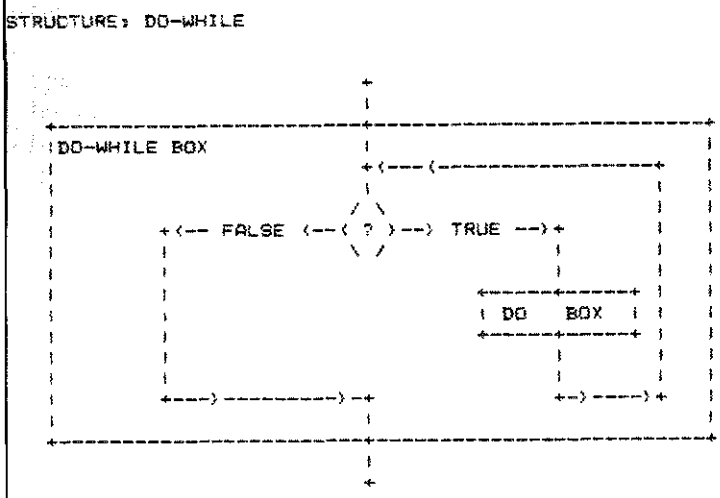
Example Conditional Expressions

```
A = 10    A<>B
A<10      A<=B     A=<B
A>10      A>=B     A=>B

SNAME$ = " Joe"
SNAME$<" Joe"
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

STRUCTURE: DO-WHILE

This consists of the following: the entrance to the structure leads to code which evaluates some condition. If the condition is true, an internal BOX of code is entered. After this BOX is executed, control returns to the decision point and the condition is re-evaluated. When the condition is false, control is passed out the bottom of the BOX.



Purpose or Use

Allows us to repeatedly execute a set of code. It makes this repeated execution subject to a prior test that determines whether it is necessary or safe to execute the set of code.

Example of BASIC Code

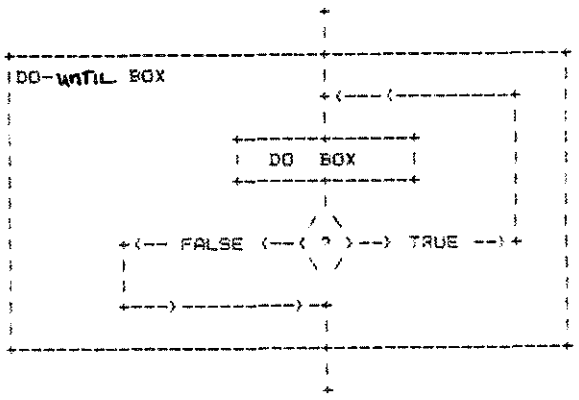
```
200     WHILE condition
210       Statement(s)
        ...
        ...
300       WEND
```

Simple Example: Delete Leading Blanks From a Character String

```
100 WHILE " " = LEFT$(LL$,1)
110 LL$ = MID$(LL$,2)
120 WEND
```

STRUCTURE: DO-UNTIL

This consists of the following: the entrance to the BOX leads to a set of code (an internal BOX) which is executed once. Upon exit form the internal BOX, a condition is tested. If the condition is true, control passes out the bottom of the DO-UNTIL box. If the condition is false, control is returned to the top of the internal BOX. Hence, we DO the Internal BOX UNTIL the condition is true.



Purpose or Use

Allows us to repeatedly execute a set of code. It is distinguished from the DO-WHILE in that the internal BOX is always executed at least once.

Example of BASIC Code

```
300 FOR variable = start TO end STEP size
310       Statement(s)
        ...
        ...
350       NEXT variable
```

WARNING: The above example was included for two reasons:

First, in many BASICs the statements from 310-350 will be executed at least once regardless of the value of start, end and step. Hence, in most BASICs, a FOR-NEXT loop is in fact a DO-UNTIL. This is a leftover from FORTRAN's DO Loop which is still Specified by ANSI Standard to be a DO-UNTIL.

Second, IBM PC BASIC does not implement the FOR-NEXT Loop as as DO-UNTIL. It is implemented as a DO-WHILE. For Example: 10 FOR I% = 1 TO N% : PRINT I% : NEXT I% will not PRINT anything if N% < 1.

(Continued next issue)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DISK EXPLORER

GOING UNDER DOS_ TO EXPLORE COPY PROTECTED DISKETTES

by Richard Larratt

One of the things it took me too long to learn is that theory is only like practice: in theory. And that when one ceases to be a self-sufficient programmer, reads the advertisements, and buys a utility to save time or whatever other rationalization one uses, one has joined the ranks of the consumers and for that sin must take the consequences.

I bought a number of the well-advertised DISK UTILITY packages. Working within the premise BAMBI gave THUMPER that "If you can't say something nice say nothing at all!" suffice to say that the NORTON UTILITIES have an excellent user interface and DISK MECHANIC will read some diskettes with copy-protect schemes i.e. such as blank track, duplicate id., etc.

DISK EXPLORER was written by Robert T. McQuaid. It will read the 'top thirty" listed in Softalk, from 1-2-3 to VisiFile. It works below DOS reading the actual contents of the disk controller and makes no DOS-like assumptions.

As delivered, it recognizes the different DOS formats up to 2.0. Also the ten-sector JFORMAT. In fact it can explore whatever is on the diskette. Bob has simply done some extra work to make it easier to work with diskettes written in these FORMATS.

The Manual is eighteen pages and fits into your IBM DOS MANUAL. The capabilities of DISK EXPLORER cover all aspects of reading diskettes.

Without listing all the capabilities, here are a few of the ones that really stand out.

FILE DIRECTORY SCREEN. You may move the cursor to any entry via the cursor control keys. You may display the file with a single <CR> after positioning the cursor on the line with the desired file.

CHANGING A SECTOR. You can alter the data of any sector of any track, including sectors which have been formatted in a bizarre fashion to frustrate this event.

To understand DISK EXPLORER you must study how data is organized on the diskette. This is not as simple as one would like because the actual data written on the diskette is a mixture of data, some structural for the system and some (what you normally see) for the user. DISK EXPLORER sees both and makes no DOS-like assumptions. All of this is detailed in the 18-page manual.

In orthodox diskettes, the C, R and N fields in the id field contain the cylinder (or track) number, the head number (0 or 1) etc. In orthodox diskettes, the value of N is the same for the entire diskette (512 on DOS 1.1) and the sectors are numbered consecutively on the track beginning from one. Not so with copy-protected diskettes. And it is only by going under DOS and working with true disk controller register contents that one can cope with the tricks of 7 sectors here, 9 there and so on.

The problem is that some of the best code is hidden behind these tricks. With DISK EXPLORER you can read & write that code. No more loading your "other" disk utility and being stopped at the onset by "READ ERROR PROBLEM" . □

TELECOMMUNICATIONS TYPESETTING

COMPUTER OWNERS: If you have a modem, text editor, and communications program, you should consider the improved readability and cost savings provided by *TYPESETTING* your program documentation, lengthy article or manuscript, newsletter, or other text instead of just reproducing it from line printer or daisy-wheel output. Computer typesetting by telephone offers you high quality, space-saving copy that creates the professional image you want! Hundreds of type styles to choose from with 8 styles and 12 sizes "on line!" And it's easy to encode your copy with our designated typesetting commands.

CONFIDENTIALITY GUARANTEED
— bonded for your protection —
PUBLICATION DESIGN, EDITING, & PRODUCTION

EDITING & DESIGN SERVICES, INC.
30 East 13th Avenue Eugene, Oregon 97401
Phone 503/683-2657

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DEFAULT VALUES AND PROGRAM VARIABLES

by Greg Estes

If you find yourself having a very pleasant experience with a piece of software, it is often interesting to sit down and force yourself to clarify WHY is was so easy to use. The "friendliness" that is alluded to is quite often a result of carefully chosen default values.

An application program can require a great many choices for the user. A good programmer will determine the most frequent choice for a question and supply the answer. The default value should also be clearly stated on the screen so that an incorrect choice is NOT picked.

A program that inconsistently chooses default values and rarely lets you know what it will do until AFTER the fact is truly frustrating. We have all had the experience of 'abuser friendly" software. The ultimate decision for what defaults are preferable lies with the end user. There are of course times when NO default value should be given and a distinct choice must be entered before continuing. The software development time spent making these decisions is of the utmost value to everyone concerned. If your users seem to be doing too many restores from backups or reprinting too many reports then you might check your handling of menu questions.

Any default value you use should:

1. Be carefully phrased.
2. Indicate the kind of answer desired.
3. Highlight the default value in some way.

The last line of the screen is normally reserved for prompt messages. If the question you are asking the user is short then you can ask the question and explain the defaults in one line. If the question is rather long you can include additional information on the message line.

Example:

Do you want summary totals ? (Y)es or (N)o :

Is the client (N)ew, (O)ld or a (T)ransfer ? :

If the possibilities are within limits then the message line can supply them all.

What BAUD rate would you like ? :

.

.

.

'Enter 300, 1200, 2400, 4800, 9600, 19200"

If the number of possibilities are more that one line can hold then you must revert to a HELP menu or use documentation. But as long as the responses are within bounds then you can stick to a message line on the bottom of the screen.

Here is what you sometimes see in a piece of software.

Do you want the Month End report or not ? :

This is sort of two questions, no default is indicated, and we are forced to go to the documentation. Not much fun.

AN EXCITING SOFTWARE DEVELOPMENT ENVIRONMENT

Perfect Software is adding exceptionally talented 'C' programmers to its software development team. Be part of a software science organization committed to developing the highest quality software. If you are extremely bright, creative and hard working, Perfect Software wants to hear from you.

Positions are available in our Berkeley, CA, Austin, TX and Eugene, OR software science facilities. Perfect Software offers opportunities in a 'C' and Unix environment. Software development tools include Unix on Vax 11/750 and related equipment.

- ☐ Porting software to foreign language versions and maintaining these
- ☐ Developing printer drivers and configuring to various personal computers
- ☐ Developing graphics capabilities
- ☐ Implementing mouse technology

Please send your resume and statement of interest in confidence to:

Perfect Software, Inc., Product Science Center
66 Club Road • Suite 250 • Eugene, OR 97401 • Attn: Gina Beyer

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For perpetually changing answers, like dates, just let them know what you have planned:

```
Account History starting from ? (Return = 7/12/83) :
```

Creatively using default values makes for quality software. It can be carried too far though if you are not careful. Only the user knows for sure.

PROGRAM VARIABLES:

Within your program you have to carry around a lot of information stored in variables. Examples:

```
IF SUM_TOTAL THEN TOT_AMT=TOT_AMT+AMT
IF NOT FEMALE AND NEW_ACCT THEN GOSUB 15000
```

If you go about naming the variables in a casual manner then you can inadvertently make some very unreadable and difficult to maintain code. You should avoid using negative style variables. Try and give descriptive names in the positive genre. Let's really gum things up and see how muddy we can make a piece of code.

Casually picking out some variable names we come up with:

```
BAD_ACCOUNT
WRONG_NUMBER
NONE_LEFT
```

As long as we only use these variables when they are TRUE then it all works out well:

```
IF BAD_ACCOUNT THEN GOSUB 5000
IF WRONG_NUMBER THEN GOSUB 6000
IF NONE_LEFT AND BAD_ACCOUNT THEN GOSUB 8000
```

The rub comes in when we have to use the FALSE designation for the variables. It forces us to mentally figure out what appears to be double negatives, and this can really get confusing.

```
IF NOT BAD_ACCOUNT AND NOT NONE_LEFT THEN GOSUB 2000
IF NOT (NONE_LEFT AND WRONG_NUMBER) THEN GOSUB 9000
```

Preferable we would choose variables in the positive descriptive sense. Like this:

```
VALID_ACCOUNT
GOOD_NUMBER
EMPTY
```

Now we can use either the TRUE or FALSE values and they are much clearer. The code reads like it would in a normal English sentence.

```
IF VALID_ACCOUNT AND GOOD_NUMBER THEN GOSUB 4000
IF NOT VALID_ACCOUNT AND EMPTY THEN GOSUB 5000
IF NOT EMPTY AND NOT VALID_ACCOUNT THEN GOSUB 6000
```

Some of the most cryptic code you will ever read is a result of using negatively named variables. It is such a seemly minor matter that greatly effects the burden of debugging a program.

There are times when it may be advisable to create a new local variable to solve some of the more complex combinations. As long as space is not an overwhelming issue and you are dealing with a large number of possibilities then you might create a couple of new ones to fill the bill for a subroutine.

```
IF VALID_ACCOUNT AND GOOD_NUMBER AND FEMALE AND NOT EMPT
        THEN NEW_RECORD=TRUE ELSE NEW_RECORD=FALSE
IF EMPTY AND MALE AND NOT VALID_ACCOUNT AND MONTH_END
        THEN DAILY_REPORT=TRUE ELSE DAILY_REPORT=FALSE
.
.
.
IF NEW_RECORD THEN GOSUB 4000 ELSE GOSUB 5000
IF DAILY_REPORT THEN GOSUB 8000 ELSE GOSUB 3000
```

If possible you should use these multiple variables as close to where they are set as you can. Make them local to the subroutine, otherwise you may find that the code is very readable sentence by sentence but you just can't find where you set up your variables. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOW TO WRITE FAST PROGRAMS

by Neil Bennett

The 80/20 rule says that 20% of the fishermen catch 80% of the fish. The 80/20 rule of computing says that control spends 80% of the time in 20% of your code.

Heisenberg's rule of computing says you can swap space for speed, and speed for space. A program can be speeded up at the expense of space, or can be made more compact at the expense of speed. An example is a software multiply routine. This can be written as 16 adds in a loop or as 16 adds one after the other. The first way is more compact but the loop control instructions take time (maybe as much time as the adds); the second is faster, but it takes up more space.

Note that this rule applies only to efficient programs; you may be able to make inefficient programs both more compact and faster. A trivial example is taking a redundant instruction out of a loop.

These two rules taken together can tell us a lot about how to write faster programs. The 20% of the code which is used over and over is called a "hot spot" (nothing like a "honkey tonk"). Some of the program just has to be there to handle initialization or special cases, but percentage-wise, it's almost never used. If you could make the 20%, (the "hot spots"), twice as fast at the expense of taking up twice the space, and the 80% half the space at the expense of halving the speed, then you would end up with a program which takes 80% of the original space and executes in 80% of the time. (On the other hand, if you choose your spots wrongly, you could make the 20% half the size and half the speed, and the 80% twice the size and twice the speed. That would give you a program which takes 170% of the original space and takes 170% of the time to execute.)

The problem is how to recognize these "hot spots" . Unfortunately they don't have a big neon sign outside. Sometimes you can tell easily. If you have a loop which is nested 4 deep then that innermost code probably gets done lots of times. Other times it is more subtle. If you have a real time system with devices causing interrupts at different priorities, you are usually grateful that it works, without wondering how to tune it. In this sort of situation, you could build instrumentation into the program to count how many times various parts of the program are executed. If you do this, you have to be careful that your choice of places to insert your probe does not influence your results. Back to the loop four deep. If you are calculating an expression which does not depend on your loop variables, such as exp(43.6), then you could calculate the expression just once at the start of the program and store it's value in a variable. It's much faster to fetch the value of the variable many times than it is to calculate exp(43.6) many times.

NEW!
powerXLIST
Debugging Tool for IBM-PC BASIC
$45°°
plus $2 p&h, o'seas $4
• Debug your BASIC programs faster.
• Helps make sense out of someone else's programs.
• Pretty-prints your programs for easier reading. Automatically indents constructs: IF..THEN..ELSE, FOR..NEXT, WHILE..WEND.
• Cross reference tables list variable names in alpha order with line numbers where they appear.
• Allows you to include variable definitions in the cross reference tables.
• Cross references subroutine entry points and other referenced line numbers.
MTP
Micro Technical Products, Inc.
123 N. Sirrine, Suite 106G
Mesa, AZ 85201 • (602) 834-0283
Visa & MC Welcome
Attractive Dealer Discounts

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

At this point, please consider the very different nature of speed and space. If a program takes 5 minutes to execute and you halve it's speed it takes 10 minutes; if you halve it's speed again it takes 20 minutes. The same thing does not necessarily apply to space. If you double the size of your program in a single user, single task system and it still fits in the machine, it has little impact on how you feel about the world. If you double the size again, and it won't fit in, then you have a problem.

All these comments only apply to compute bound jobs. If you have a program that is spending most of it's time waiting for you to type numbers in through the keyboard, then probably a course in typing would help. If the program is doing lots of floppy disk I/O then a hard disk may be the way to go. However, you must look at each situation afresh. Years ago, I was tutoring computer science students and every one of them thought that disk was faster than magnetic tape. I long forget the figures, but here's an example where tape is faster. Suppose you are reading short records and doing some computing, and then reading the next record. On disk, the record you want has just passed under the read head and you have to wait a complete revolution of the disk before you can get to it. On tape, you don't have any problem. [There are obvious remedies to the situation, such as buffering a complete track of the disk in memory]

You may not be interested in trying to speed up a program. If you would spend more time making it faster than you would save overall, then it's not worth it. If you are going to execute the program many times, or speed will help your program compete in the market-place, it may be worth it. If the execution time is going to be about 5 minutes and you want to go and watch the football, you could just let it run. If you have a one time calculation and it's going to take 6 hours, you could run it overnight.

Usually the most dramatic increase in speed comes from solving a problem in a much better way. There is an old problem about two people on bicycles two miles apart heading towards each other. One is going at 8 miles/hour and the other at 12. A wasp which could fly at 60 miles/hour started from one rider and flew towards the other. When the wasp got there, it turned around and headed back; and flew back and forth until the two riders passed each other. How far did the wasp fly? There are two ways of solving this problem. The obvious way is to calculate the distance the wasp flew till it met the other rider, then calculate the second leg, etc. and add them all up. The other way of doing it is to consider that the two riders are closing at 20 miles/hour, the wasp can fly three times that speed, so while the riders cover two miles, the wasp will cover 6 miles. Look at the difference in the arithmetic involved! □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOSTCOMM

by Jim Bumpas

HOSTCOMM
price $170
NF Systems Ltd.
Box 76363
Atlanta, GA 30358

Hostcomm from NF Systems may be the most widely used 'bulletin board system'' (BBS) for the PC. A ''BBS'' is more than just another piece of communications software. With such a system one may leave a computer unattended and access it from a remote computer or terminal, using a telephone and modem. Some terminal programs, such as 'Crosstalk'' by Microstuff, also support unattended remote operation for such tasks as uploading and downloading files. The BBS adds the ability to deliver and receive messages (electronic mail) to and from callers.

HOSTCOMM automatically searches the message files for messages addressed to the caller and will tell you as soon as you ''log on'' if there are any messages for you. This system also organizes download files into 9 different 'directories'' Files may be placed into a directory by the 'SYSOP'' (System Operator) or by users (callers).

Users may be assigned passwords to limit access to menu items and download directories. Different levels of access can be assigned to various passwords for varying combinations of access to menus and directories.

The Sysop has access to a menu from which he can influence the performance of the system even while a caller is on line. He can force a chat with a User while the User is trying to access some part of the BBS. The User, on the other hand, can only page the Sysop, and the Sysop can turn off the bell to ignore the page. The Sysop's menu permits granting temporary access to a User without compromising the security of the normal password and access system. The Sysop menu also includes other operations and maintenance procedures which include turning printer on/off, list passwords and their access and directory assignments, view the download directory, starting and restarting the system, and return to DOS (closing files).

The Sysop's menu also accesses a Mailbox menu. This menu permits maintenance to a significant feature of the program. Messages to single or multiple addressees may be put into the BBS. Bulletins to all users may be put here. The program permits listing the text to the CRT for proofing. Individual lines may be edited. Messages can be saved or deleted. Hard copies of any message or the directory of messages can be printed out. Messages ''used'' (delivered to addressees) can be automatically purged by the Sysop. As the directory may hold only 50 addressees, the documentation advises periodic use of this feature.

Another feature not common to BBS is the ability for users with the applicable access level to run programs from a remote terminal. The ''Special Feature Programs'' are limited in number to a maximum of three. And the BBS requires the Sysop to provide sufficient coding to fit the BASIC program into the ''shell'' which is included with the BBS package. For long programs I think this coding required will be quite extensive, so this feature may not be very practical for other than short programs. You cannot just put a program into the Special Feature Program menu and let it run from there. The documentation suggests a Special Feature Program written in Compiled BASIC or is a Command program handling its own communications (these must have '.EXE'' or ''.COM'' extenders) may be used without any of the fuss required for Interpreter BASIC programs. But I couldn't find any discussion about how to set up such a program for running in the documentation. Does one merely add the program to the Special Feature Program menu? The ones I tried with this method did not function.

The program is easy to use. Once you have all the parameters established and your ''SETUP.DAT'' file created, the program is virtually self-executing. Power up with the system disk in, and off it goes. If power is lost during operation at any point, the system automatically re-boots. It asks the first user after such a power failure what the date and time is.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

When I first began to use the program, I became confused over which disk was which. The documentation refers to the two disks (at different times) as "Disk #1" and "2" ; 'System Disk" and "Scratch Disk" ; and, "Red Dot" and "Blue Dot" . It was a little difficult for me to determine, at first, just which files need to be on one disk or the other for the BBS to operate. While using "Install.exe" to create my SETUP.DAT file, I encountered an "Unexpected Run Time Error" several times. I have no idea what this error means. It occurred when I was using the Screen Shell to set parameters for the BBS operation. I found it easier to create the SETUP.DAT with the Prompt Line Edit mode of the Install program. This feature is designed to be used to change system messages and prompts.

You need to be careful when using this Install program. Once SETUP.DAT is created, check it to see if it contains all you intend it should contain. I was careless on a couple of occasions and was unable to use the BBS from a remote location. Now I am comfortable with the system, and find it very convenient. My partner and I can leave messages for each other on it without using up the tape on the answering machine (on the other line). We can send files back and forth from home (or other locations) to work on them when we're not in the office. My partner has a PC at home. I only have an Atari, but I've experienced no problems working with files on my word processor at home and then sending them back to the office. It's almost like having another employee working at the office round-the-clock to handle files and messages. HOSTCOMM is an excellent BBS. The telecommunications use of micros is proving to be one of the most explosive growth areas as more and more people set up their own BBS, or use their micros to access BBS set up by others.

I have one suggested improvement. In my experience, it is a good idea for users to be able to upload a file into a private BBS which cannot subsequently be downloaded by another user without the intervention of the SYSOP. To accomplish this, there needs to be a menu option by which the BBS creates an extender to the filespec which will prevent it from being accessible to a download directory. □

THE PROGRAMMER - A REVIEW

by Ron Moak
THE PROGRAMMER - A Program Generator
price $199.95
Howard W. Sams & Co. Inc.
4300 West 62nd St. P.O. Box 7092
Indianapolis, IN 46206

INTRODUCTION

Ever since the beginning of the computer age, programmers have been striving to the ease the task of programming computers. Assembly language mnemonics have replaced binary code, and high level languages have replaced assembly language. After thirty years, the process of writing and debugging computer programs is still long and complicated. In an effort to reduce that process, and open up computer programming to people with neither the desire nor interest to learn a complex programming language, The Programmer was developed. The Programmer is a software tool designed first to allow people to develop complex programs without having to learn a programming language, and second to increase the efficiency of programmers.

The Programmer accomplishes its task by generating code as you step through a series of menus selecting what you want the program to accomplish, and interjecting variable names, parameters, and comments where needed. For example, if you wish to create a random access file, you select file creation option under file control menu under the Main menu and enter the names of the variables, their length and type and the program generates all of the code necessary to generate the file. Quite complex programs may be written with The Programmer, and many of the functions that we like to use, but often leave out, such a checking for input ranges on numeric input, are automatically included in the generated program.

PROGRAMMING ENVIRONMENT

The program opens with a list of attributes of the program you are currently working on (The Programmer assumes you will work on only one program per disk). These attributes include printer width, current line number, program name, etc,. You may change the attributes or continue on with the program development. You are next prompted to include information on the files your program needs. If your program does not use files, you may bypass this section by hitting enter.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Following program and file setup, you are presented with the first of a series of menus. The Programmer divides program development into the eight areas (file control, printer control, console control, program control, alpha functions, numeric functions, and special functions) listed in the "The Main Systems Menu". From this menu you have access to the submenus that cover a particular aspect of the computer program. Each area of emphasis contains all of the commands needed by the programmer to perform a designated task. For example, the commands covered by program control include all branching, repeating delays, halts, calls, chaining, remarks, and BASIC statements.

Items in the submenus may lead to further menus depending upon the complexity of the task. When no more menus are needed to subdivide the problem, the program prompts you to enter the needed information. The prompt for a branch asks for the line number to branch to and if the branch is to a subroutine. Other prompts ask for variable names, variable types, comments, etc. Generally the types of information called for are parameters for the BASIC statements.

If the program does not provide a ready made solution to a problem, you may enter straight BASIC code; however, you are warned that the syntax of the statements you provide is not checked. Most of the second level menus provide an access to generating straight BASIC code.

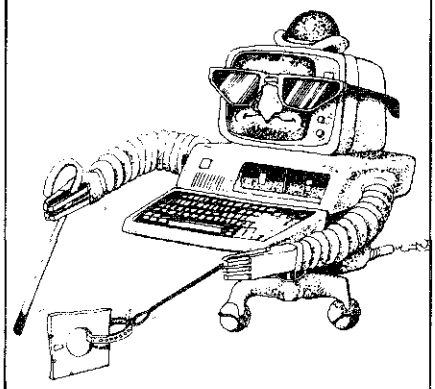
In addition to the areas of programming control, the main menu provides access to the system utilities. The utilities allow you list the program on the screen or printer, retype a line, delete a line, or check the program length in bytes. You may also select the comment option that will prompt you after each line generated to enter a comment.

When you have finished entering your program, The Programmer assembles all the program modules (new modules are created in each editing session), plus additional code The Programmer adds for initialization, into a working program. Once the program is assembled, you may copy it to another disk and run it like any other BASIC program.

DOCUMENTATION

The Programmer comes with a manual in a 8 ½ by 11 notebook that may be stood up like an easel, and a disk. The manual is divided into a 53 page primer, a thirty page reference manual and the appendices. The primer provides an overview of how the program works. An

We can help your IBM C.



Powerful, portable and fast, C is the language of tomorrow!

With Computer Innovations' C86 compiler you'll have full unix-compatible C running on your PC or other 8086/8088 system. C86 is a complete implementation, with struct/union, bit fields, trig functions and floating point with 8087 support.

The source code for the run-time library is included — you pay no royalties on programs you produce!

C86 is only $389, or $399 with K&R "The C Programming Language" reference manual. And with each order we include a free copy of our powerful modem/file transfer program VaMP, a $100 value!

Call us toll free at 1-800-426-5248. We'll help your PC "C" the light!

VISA, MasterCard and AMEX accepted.

VANDATA

For more information call: (206) 542-7611 or write Vandata: 17544 Midvale Ave. N., Suite 107 Seattle, WA 98133

unix is a trademark of Bell Laboratories.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

example program is included that takes you step by step through the process of entering a program. The primer provides a good overview of random access and sequential file structure and data types. An index is included, however, the two times I looked for something I could not find it. The disk contains the program files (written in BASICA and not copy protected) and data files.

COMMENTS

The real power behind The Programmer is the ease in which it allows programmers to develop programs that make extensive use of files. The example program provided to with The Programmer, generates a mailing list using a random access file that allows you to enter new records change old records, print records or list the records on the screen. This complex a program is not what you would normally launch into with a beginning programmer. Yet The Programmer handled it fairly easily.

In the course of working with The Programmer I have developed a number of reservations concerning the overall usefulness of this program. Whether my reservations have any meaning to you depends chiefly upon your applications and tolerance. While the theory behind The Programmer is good, in practice the program falls short of expectation.

As you have probably determined, this program has many menus. In fact there are twenty different menus. As a matter of opinion, I take issue with the feeling of many of software writers who seem to feel that the more menus the more user friendly the program. While an occasional menu is more comforting than a blank screen when you are completely lost, too many menus and levels simply confuses the user. In the end it becomes more difficult to use the program.

In The Programmer, menus may not be bypassed. This requires you sit and wait for the each menu to be displayed on the screen before continuing on to the next. To make the matter worse, after you have worked your way down to a level, you have to work your way back out to the master menu with repeated pressing of enter key. As a final blow, every time a menu is changed, the screen is completely rewritten even, though only a few lines in the middle of the screen are changed. A faster and more efficient method of movement around in the program is needed for experienced users.

To enter a simple goto statement from the main menu you must go through the following steps.

User	Program
Select four	clear screen display program control menu
Select 1 (branching)	clear screen display branching menu,
Select 1 (branch unconditional)	display prompt "Enter Line number"
Enter line number	display prompt "Branching to subroutine (Y/N) ?,
Answer question	clear screen, display branching menu (menu was already on screen)
Press enter	clear screen, display program control menu
Press enter	clear screen, display main menu

This simple series of operation required five separate screen rewrites. After a couple of hours of being bombarded with screen clears, I rewrote the source code to clear only the middle of the screen. It seems utterly ridiculous to spend two hundred dollars on a program only to have to rewrite it to make it more efficient.

One of the strong selling points of the program is the ability to program without knowing a programming language. This is true, provided your program logic is perfect when you enter the program and you make no mistakes spelling variable names or entering numbers. The program does not check the validity of your input as far a I could tell. One program I wrote aborted because I had attempted to GOTO a line number that contained character data. The program should at least prevent you from entering character data where it prompts you for numeric data.

The program makes a low beep to let you know it is writing code; however, no record is kept of your keystrokes to keep track of what you have entered. Your only frame of reference is a number in the corner that keeps track of the number of steps taken and a listing of the current line number of the program. If you make a mistake there is no way to trace your steps without listing the program. This is fine if you have a good background in BASIC, but the whole idea in the first place was not to have to know the language. In most cases you cannot tell you entered something wrong until after the program is assembled and you attempt to execute it.

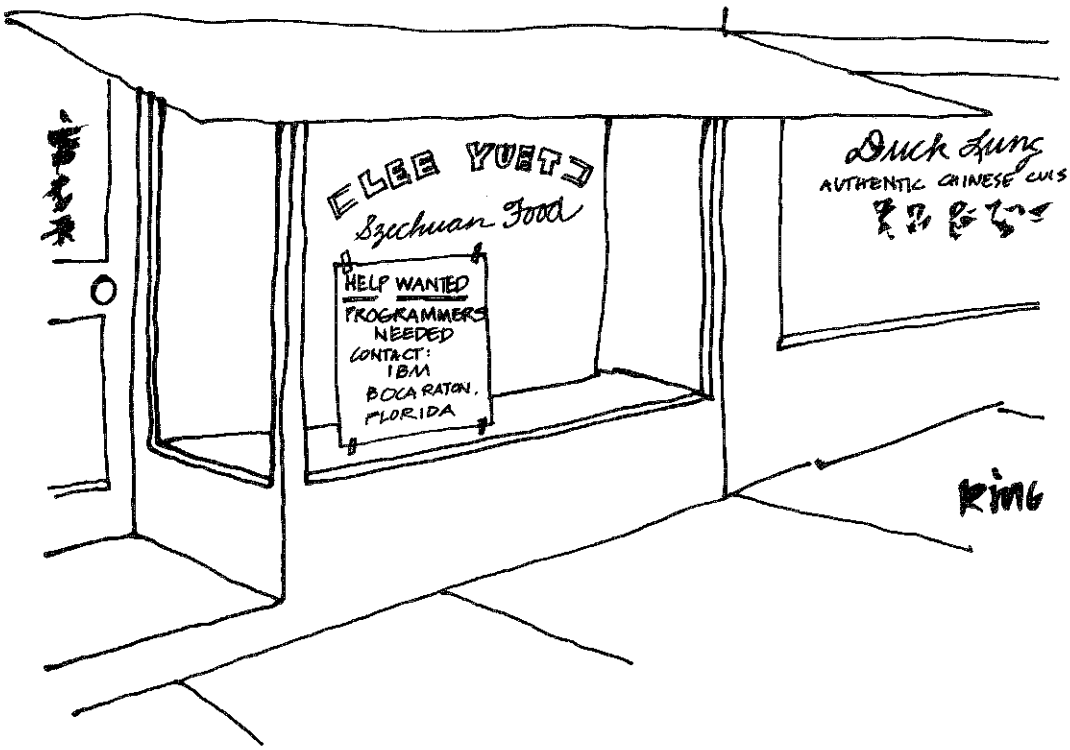
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

My first attempt to run the practice program resulted in a illegal function message and program abort. When I listed the problem line, I was greeted with a 230 character line composed of compressed programming containing multiple statements with one character variables. It turned out that I had misspelled a variable later in the program and one of the variables in that statement was not initialized. I know I would not have been able to able to debug the program if I did not know BASIC.

The real trouble with this program lies in the fact that you are programming on one level and having to come back and debug on another level. It is like writing a program in BASIC and then debugging it in assembler. The program does offer some utilities for modifying the program but, in my opinion, they are of little value. Correction involves rekeying the correct BASIC code. In order to edit the line you must list the program, determine the number of the bad line, return to the menu, and then request to edit the line. You cannot edit the line in reference to others around it with out a hard copy of the program. This kind of editing requires knowledge of BASIC.

SUMMARY

If you have plenty of money and tolerance, perhaps this program will find a place in your software library. My advice is to wait until something better comes along. □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE RANDOM HOUSE PROOFREADER

by David Whitman

Random House Proofreader
price $50
Box 339-M
Tijeras, NM 87059
ph. 505-281-1634

It would appear that Aspen Software is out to take the spelling checker market by storm. Proofreader is a big league spelling program, with a little league price. At only $50, the program costs only a fraction of the price of most of the other spelling software on the market. Despite the price discrepancy, Proofreader is reasonably fast, and has some very useful features not found in many such programs.

The Package:

Proofreader comes on a single disk, with an 18 page documentation booklet. The booklet appears to be xeroxed dot matrix printer output, but is quite readable. A table of contents is provided, but no index. The documentation is well written, and contained all the information I needed to use and understand the working of the program.

I am delighted to report that the disk is not copy protected. Aspen recommends that you copy the program and its subsidiary files onto the same disk as your word processor, for convenience. This is welcome change from the many packages in which more effort appears to have been expended in devising a copy protection scheme than in writing the program itself.

The dictionary file which accompanies the program comes in several sizes, to suit your disk capacity. For double sided disks, a lexicon of 50,000 words is provided, which makes the total package take up about 160K of disk. Single sided disks come with a lexicon of 32,000 words and take up 120K. If you upgrade to larger floppies or a hard disk, the manual states that Aspen will provide larger lexicons for "a nominal disk media charge"

The lexicon supplied with Proofreader is derived from the Random House Concise dictionary. The documentation describes major effort which went into generate the lexicon and insure its accuracy. I like the idea of having a big name dictionary stand behind the program, since a spelling checker whose lexicon contains errors isn't worth the disk it's distributed on.

The program requires 64K of memory and one disk drive, although two disk drives make life much simpler. The documentation claims that the program is compatible with 'almost any word processor or text formatter that runs on the IBM Personal Computer" . I tested the program with files produced by Easywriter 1.1 and EDLIN, and for the most part, I can confirm this claim. I had a few problems with Easywriter files, which will be mentioned below. The program has special modes for use with Wordstar, so presumably it works there as well.

Running Proofreader:

Checking a document with Proofreader is done in two phases. During the first phase, Proofreader scans the file, building up a sorted list of all the unique words. The word list is then compared against Proofreader's dictionary.

The comparison takes a while. For the 50,000 word dictionary, the comparison takes about a minute and a half, and is independent of the size of your text file. During that time, the disk drive runs continuously. It would appear that the entire dictionary is being read and matched against the unique word list, each unique word being crossed out as a match is found. It seems hard to imagine that this is the most efficient way to do the comparison, although I'm no expert in this field.

A line of letters is maintained on the screen during the search, each letter being crossed off as that portion of the dictionary is completed. The documentation claims that the process is faster with a hard disk, which is consistent with an I/O bound program.

The second phase involves correcting your document. This is where Proofreader really shines. Many spelling checkers mark misspelled words in your document with a special character, which you then search for with your word processor to correct. Although you are given the option of marking words for later correction, Proofreader's interactive correction mode is so good that I can't imagine wanting to do so.

During the correction phase, Proofreader scans your document again, stopping for each word which was not found in the dictionary on the first pass. At the density of errors which I typically generate, the delay while each misspelled word is found is quite short, about a second.

Each word is displayed on the screen in context, and you are asked for a response. Your options include correcting the word, adding it to the dictionary, or accepting it as correct without adding it. These are pretty standard options for an interactive spelling checker.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

What isn't standard is the dictionary lookup option. Upon request, Proofreader will display a portion of its dictionary on the screen, to let you look for a correct spelling. This is incredibly useful. I have used other such programs on a mainframe computer, which ran a little faster, but I spent all my time balancing a dictionary on my knees, fumbling to look up the words I'd misspelled. Having access to the on-line dictionary makes the total process of finding errors and correcting them to be much more efficient.

There's not much more to say about using the program. It's convenient, and it works well. Those are the things I look for in a program.

Problems:

I discovered only a few annoyances with the program. I wish the dictionary scanning phase were a little quicker. The 90 second delay, regardless of document size, tempts one to skip checking a short letter or memo.

When you correct a word, Proofreader then checks your correction, which is great. However, if the corrected word is not in the dictionary, you aren't given the option of adding it. You really should have all the options available at this point that you do with the original misspelling.

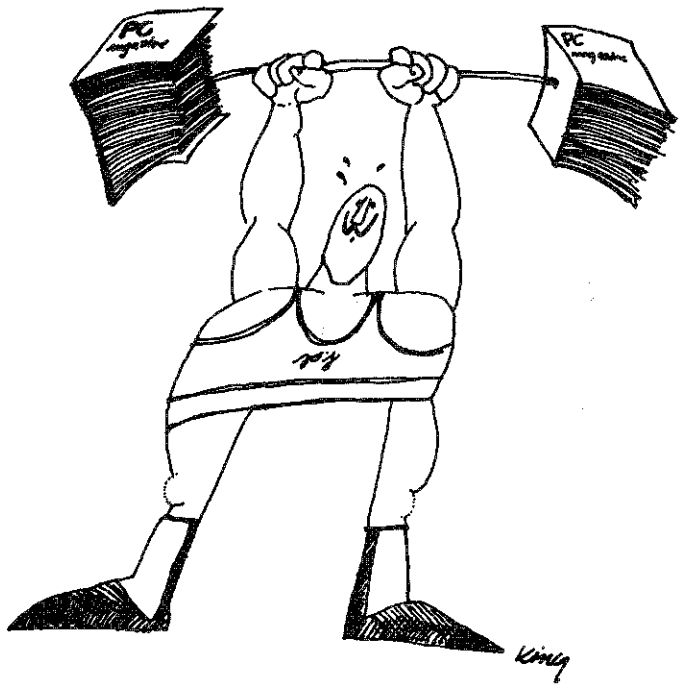
The biggest problem is with Easywriter compatibility. I find it hard to place the blame here on the writers of Proofreader, but they are claiming compatibility.

Very occasionally, Proofreader will read an Easywriter file, and come up with a list of zero (0) unique words. The program doesn't crash, it just seems to have trouble reading the file. The problem is the header which Easywriter adds at the beginning of each file. Easywriter stores such things as file size, and tab settings in the header. The files which caused trouble were those in which the file size in hexadecimal contained the digits 1A. Unfortunately, DOS reserves character 1A as its end-of-file marker. When Proofreader gets to the 1A, it just stops, thinking that it's done. I confirmed this by taking a file which worked OK with Proofreader, and setting a tab stop at column 26, which corresponds to hex 1A. After revising the file, Proofreader couldn't read it, either, presumably because Easywriter put a 1A in the header to record the tab.

You can get around the problem by converting the Easywriter file to standard ASCII format using the TRANSFER utility, but this is a bit of a pain. I think the real blame for this problem belongs to Easywriter. The header information could just have easily been recorded at the end of files, allowing other programs to access the text and ignore the extra data. Letting an end-of-file slip in before the text is inexcusable.

Conclusion:

The Random House Proofreader is a highly useful program for locating and correcting spelling errors in text files. The combination of low cost and ease of use makes the program a good value for anyone doing word processing on the IBM Personal Computer. □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

REPORT TIME

by Frankie Detesto

A good (and very tired) friend of mine just completed scoring a large marathon race using dBASE II. He was in charge of all the runners' entry information as well as the race results. This involved over 4,000 runners.

There was one master data file which contained everything from a runner's name and address to their race number, T-shirt size, age, sex and so on. The file ALMOST fit on one double sided diskette. Not too bad for a fair sized marathon. The squeeze came when it was time to do all of the printouts for the results.

In the heat of the data processing there came an all too familiar request that is close to the heart of programmers everywhere, "We want the results NOW, thank you."

As I leaned over his shoulder and noticed the commands he was typing in I decided to just write them down and pass them on to you in their "SET RAW ON" form. Here are some of the ones that succeeded in the midst of chaos.

A large index file was created to quickly access the information in whatever order the officials wanted. It was discovered that doing this to a RAMDISK decreased the index creation time.

```
USE file1
INDEX ON field1 TO C:index1
```

One of the big boondoggles was allowing runners to enter immediately before the race. The problem becomes how to printout reports for the last 200 runners you added. This can readily be done with the REPORT and FIND and NEXT statements.

```
USE file1 INDEX C:index1
FIND LASTNAME = "MORRISON"
REPORT NEXT 200 FORM report1 FOR SEX = "M"
                                    TO PRINT
```

Sitting with several users around you who are waiting for their next printout can be a little nerve-racking. This next set of commands was a creative attempt to solve an APPEND problem, while thinking privately "if I could just go home and get the manual I know I could do it easier."

The APPEND command refused to execute from one DBF to the other so the SDF (System Data File) format was used.

```
USE file1
COPY TO file1 SDF FOR SEX = "M"
USE file2
APPEND FROM file1 SDF
```

In other words, take a regular dBASE file and make it a standard ASC II file and then append it to another DBF file. I never even thought of doing it this way.

The last obstacle course for that marathon was to take the main file (which was spread over 2 diskettes) and extract just the minimal fields so it would all fit on one diskette. This was necessary to figure the final finishing order for the runners as well as divide up prizes.

Basically you have to copy the main structure over to another file. Take out some fields to make it smaller, and then APPEND just the fields you want from the split main file.

```
USE file1
COPY STRUCTURE TO file2
USE file2
MODIFY STRUCTURE (take out some fields)
USE file1
COPY TO file2 FIELD field1, field2, field8, field10
  FOR SEX = "M"
```

Often our most creative efforts surface when we are forced to do the best we can in the situation at hand. □

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ LIBRARY

PJ LIBRARY

PCA & DEVICE DRIVERS

The installable device driver capability of DOS 2.0 is quite a nice feature. Personal Computer Age has an excellent article on this subject by Barry Crawford. It is in the Vol 2.6 issue of PCA. Both a discussion of the subject and some sample ASM code are included to illustrate the process.

PC MONITOR DOS IN COLOR

The June 83 addition of the PC Monitor newsletter has a very extensive ASM listing of how to use color while in DOS. You can set the foreground and background,etc. Also included are some DEBUG tricks to set up the Personal Editor so that it works in color. The ASM code for DOS color is extensively commented and highly tutorial. To get a copy write to :

CAPITOL PC MONITOR
13700 Beauwick Court
Silver Spring, MD 20906

ASSEMBLER BOOKS

There are 2 new 8088 Assembler books on the market.

8088 ASSEMBLER LANGUAGE PROGRAMMING: IBM PC
by Willen & Krantz
price $15.95
Howard W. Sams & Co., Inc.
4300 West 62nd St.
Indianapolis, IN 46268

This book was written by 2 electrical engineers from New York. It is very thorough. There are quite a number of illustrations which clarify bit movement, flags, etc. An overview of 8088 architecture is discussed with an eye toward the eventual programming. Specific chapters cover BIOS-DOS-MASM, 8259-8255-8253 programming, Mono-Color-6845 Controller, Serial Communications (8250), and Disk I/O. Each major chapter includes an ASM program to illustrate the material covered. The book is superbly laid out and the text is well-paced. If you have done some programming and wish to read a lucid explanation of the major considerations involved with assembly language then you would do well do get a copy. We had not seen this book until the PC 83 Faire and several other ASM programmers recommended it highly. It is refreshing to find such a graphically pleasing book, written by experts on the subject that is for programmers.

IBM PC ASSEMBLY LANGUAGE:
A GUIDE FOR PROGRAMMERS
by Leo J. Scanlon
price $19.95
Robert J. Brady
Bowie, MD 20715

This is the book heavily advertised in PC Magazine and elsewhere. It covers MASM, 8088 Instruction set, High precision mathematics, Data structures, BIOS-DOS-I/O, and Graphics and Sound. There is a great deal of information in this one. The layout of the book is a little too list like. Here is an instruction, here is what it does, here is a short piece of code to look at. If you have a copy of the 8086 manual and wish to have the instruction set elaborated on then this book would be a reasonable choice. Also if you are interested in mathematical or sort techniques then it would be a definite help. Several 32-bit X 32-bit, Division, and Signed number example programs are covered. The usual Bubble Sort and Ordered List manipulations are explained and illustrated. If you have a graduate R&D inclination for using the assembler language then this book would certainly give you a boost forward.

PETER NORTON

INSIDE THE IBM PC: ACCESS TO ADVANCED FEATURES AND PROGRAMMING.
price $19.95
Peter Norton
2210 Wilshire Blvd. #186G
Santa Monica, CA 90403

If you have enjoyed Peter's articles from Softalk and PC Magazine then you will no doubt enjoy his book. He has a distinctive style which is an enjoyable break from the dry Tech manual approach used in most computer books. The title accurately describes what the book succeeds in doing, giving you an in depth understanding of what the IBM PC is all about. Everything from DOS to ROM to Diskettes to Graphics is mentioned. It is a guided tour of our machine by an articulate professional. The only caveat is the use of Pascal in contrast to only a sprinkling of BASIC and ASM. As the book is a tour of the IBM PC, having that much Pascal code detracted a little. Peter would do well to create a Pascal Tools book and diskette to satisfy the interest there. This is only a minor point about an extremely important tutorial book.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

(An aside: Both the Scanlon and Norton book are published by Brady & Co. There is one aspect of the 2 books that seems out of place, and it has NOTHING to do with their content. The cover art and the internal layout of the books seems adolescent at best. Considering that you will be paying $20 for either book they could obviously have afforded a talented cover artist or photographer to do the outside. The insides of the book reflect a "make a quick buck" approach to the whole works. Fat pages with a muddy typeface and minimal layout design. Considering the prestige of Peter Norton and the cost of the book, we should have been treated better. Brady should take a look at the Howard Sams ASM offering and see how nicely things can be put together for $5 less per book.)

PC TECH JOURNAL

The first issue of the PC Tech Journal is finally out. It is edited by Will Fastie so we should see some nice features as time goes on. The first issue had an excellent article on "How to choose a C compiler" by Bill Hunt. Very thorough. Also a complete guide to graphics programming for the Color adapter is included. Will is still kicking sand in IBM's face about their computer design. Other articles contain a review of the Norton Utilities, a new rationale precompiler for BASIC, 3-D images, a benchmark discussion and so on. There is an obviously talented Art department designing the Tech Journal. Gerry Gersten's drawings are superb.

SOFTALK

The July issue of Softalk has a clear discussion of the 'snow" problem that we have all seen on various color monitors. It is in the Q&A section on page 17. This explanation alone is worth tracking down a copy. This issue all explains how BASIC stores variables and mentions MS-DOS 3.0 and alludes to what PC-DOS 3.0 might contain.

BYTE

The July issue of Byte has several interesting IBM PC related articles. Dan Rollins has written a very complete article on calling ASM subroutines from BASIC. If you have been having trouble doing this or want to understand the mechanics better then you should read the article. Also in this issue is an article on 8086 vs. 8088 benchmarks, the 8088 does quite well.

INTERNATIONAL PC OWNERS

IPCO

"The international user group for the PC"

- Software library of 175+ programs @ $3 each
- Datacount® General Ledger/Manual $35
- Volkswriter® $125
- Bimonthly newsletter 40+ pages
- Worldwide information exchange
- "Freeware"

FOR FREE CATALOG WRITE:

INTERNATIONAL PC OWNERS, INC.
ATTN: FREE CATALOG
P.O. BOX 10426
PITTSBURGH, PA 15234

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MISC4

VISICALC MEETS LOTUS 123

If you have been using VisiCalc for some time and a new product like LOTUS 123 comes out then your options are either to stick with the old model spreadsheet and eat your heart out, or fork over the money for the new style spreadsheet and go through the laborious process of learning a new set of commands. But VisiCalc user have now been saved by the Multisoft Corp. They have an add-on package for VisiCalc that gives you integrated graphics, function key definitions, and sorting/rearranging of your spreadsheet information.

The program is called StretchCalc and is loaded along with VisiCalc in memory. One of the best parts of the whole package is never having to learn a complete new set of commands. You remain in VisiCalc the whole time. You can take your information and graph it 8 different ways, by lines or bars or pie charts, etc.

StretchCalc is very professionally implemented as it is written by a talented group of experienced programmers. Have the best of both worlds with StretchCalc and VisiCalc. For $99 it is quite extensive as well as easy to use. For more information contact your local dealer or call:

Multisoft Corp
14025 S.W. Farmington Road
Beaverton, OR 97005
ph 800-322-4110

USUS

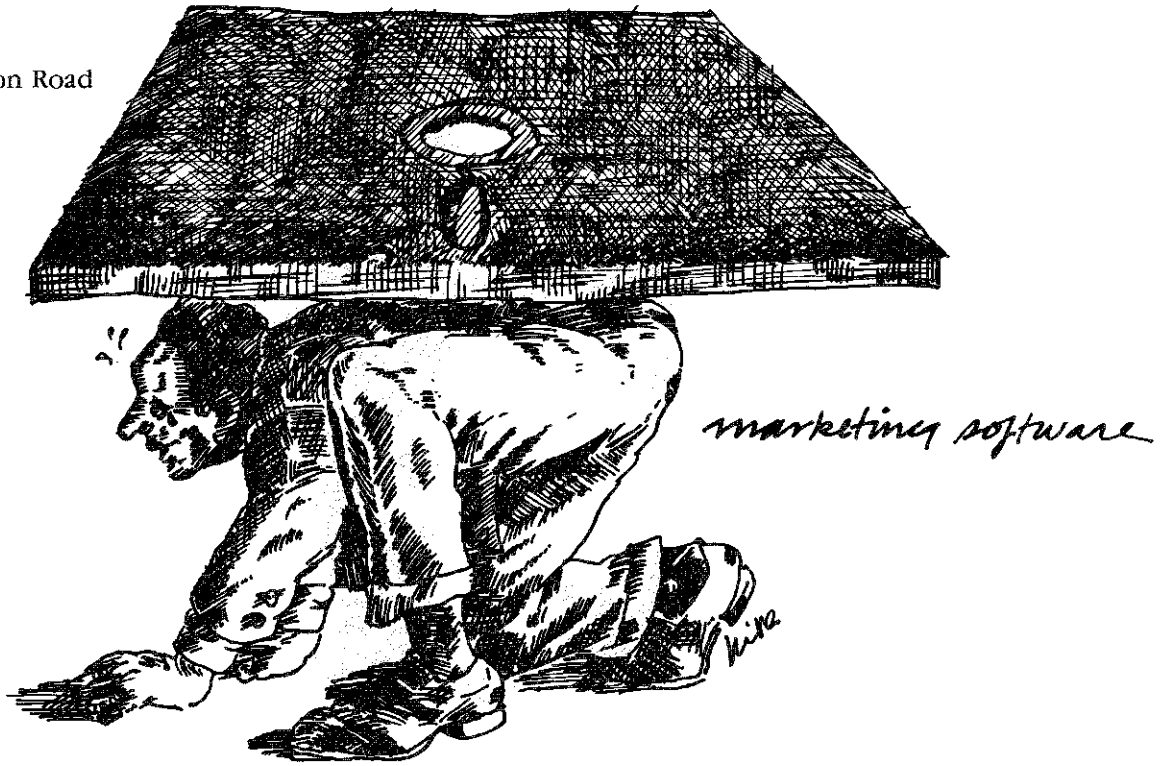
There is a new UCSD p-System user group called USUS. They desire members from every type of computer user that has the p-System. Contact:

USUS
P.O. Box 1148
La Jolla, CA 92038

FORTRAN USERS MATH PACKAGE

If you use FORTRAN and want to vastly increase your math speed with an 8087 chip then REALFAST may be of interest to you. It speeds up large integer and real number computation by directly accessing the 8087 chip. Works with any version of DOS. Contact:

Realfast
Geostat Systems Onc
10403 West Colfax, Suite 450
Lakewood, CO 80215
ph 303-277-0070



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SRITEK CHIPS GALORE

This company has a number of CPU chips that you can use in your IBM PC by using their VersaCard. Basically they have an expansion board (VersaCard) which can contain up to 256K of memory or 512K with a piggy back module. Using this VersaCard you can then plug in the appropriate CPU you would like. The full spectrum of current chips is well covered. Take your pick of an 8086, 68000, iAPX-286, 16032, or Z-80. This allows you to run UNIX, XENIX, and MP/M. Make you IBM PC look like a VAX. For information and pricing write to:

SRITEK Inc
10230 Brecksville Rd.
Cleveland, OH 44141
ph. 216-526-9433

MICRO SOFTWARE DISTRIBUTORS

This will be a new directory listing companies that market and distribute software for you. Each company (160 in all) will have a description of the kind of products it is looking for as well as the agreements they desire. The book is $95 and should be out in late July. Contact:

Software Research Co.
P.O. Box 9524
Washington, DC 20016
ph. 202-364-8700

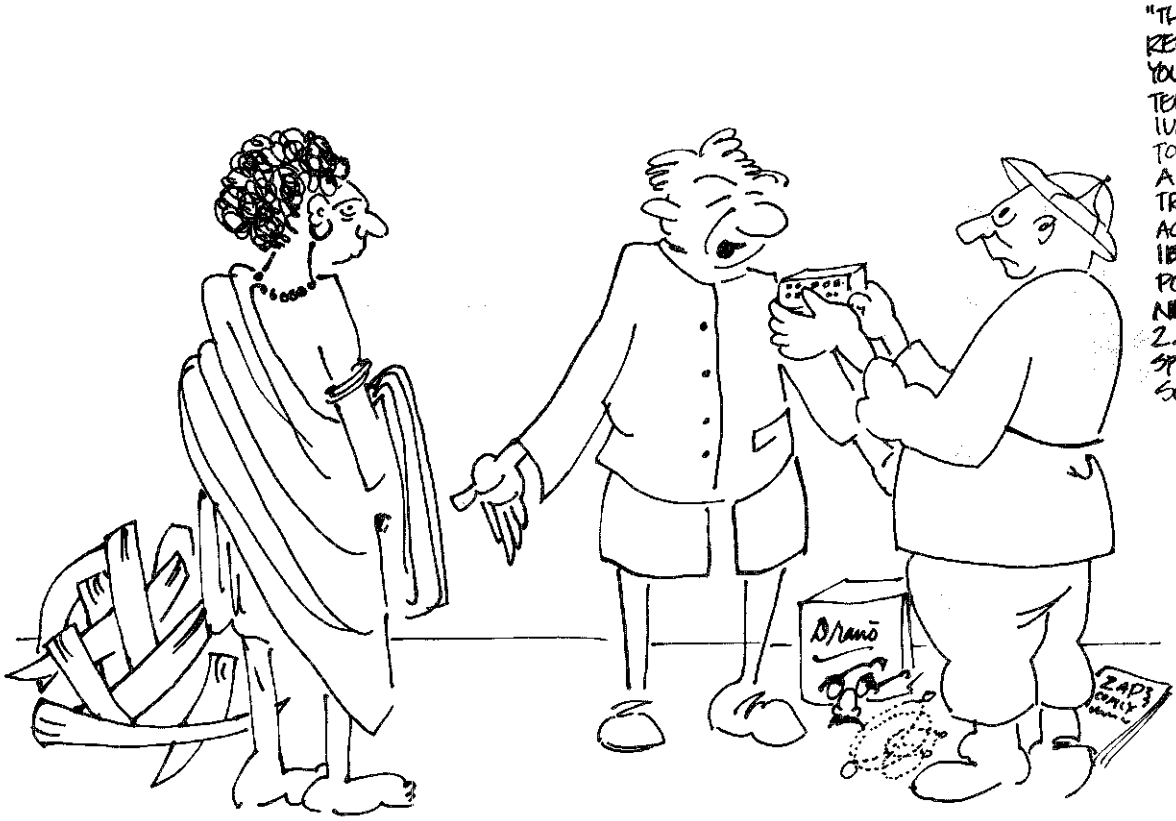
MONOCHROME EXTENSION CABLE

This note form the Portland IBM PC Users group. You will notice that there are a number of "extension cable" products out for IBM PC. Most of them mention the keyboard but not so many take care of the shortest of all cables, the AC power cord to the Monochrome display. Well just skip all of the ad hype and go directly to your local electronic supply store and order BELDEN part #17524. This is a pre-made extension cord for the Monochrome display and costs only $2.54. The cable will work with the Curtis Cable and costs considerable less.

IBM PC REACHES CICS

There is a product that hooks you intelligently to your OS or DOS system with CICS. Any databases supported by CICS can be downloaded to your IBM PC. If you are thinking about using a mainframe-based communications package to link up IBM PC's, standard IBM Terminals and word-processors then OMNILINK may be of interest to you. Prices start at $25,000. Contact:

OSI
Fort Lee Executive Park
2 Executive Dr.
Fort Lee, NJ 07024


"THE EXALTED TRIBAL CHIEF RESPECTFULLY ACKNOWLEDGES YOUR OFFER OF A VIC COMPUTER IN EXCHANGE FOR HIS LUDRY TUSKS; HOWEVER, DUE TO HIS POSITION AS SUPREME AFRICAN MULTI-NATIONAL TRIBAL COMMANDER, HE CAN ACCEPT NO LESS THAN AN IBM PC XT WITH A FULLY POPULATED MEGAPLUS BOARD, NEC 3550 PRINTER, DOS 2.0, AND COMBINATION SPREADSHEET, GRAPHICS SOFTWARE."
Drano
ZAP! COMIX

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BELLSOFT: PASCAL & C TEXT EDITOR

A new firm in Seattle is publishing a Pascal Program Editor that is designed to help you write your Pascal (and C) programs. It checks your syntax as you type in the text as well as produces the structure and key-words for you also. A definite advantage if you are having trouble with your Pascal syntax or are new to the language. Their C version will be out later this summer. For more information contact:

BELLSOFT, Inc
2127 Bellevue Way, S.E.
Bellevue, WA 98004
ph. 206-451-9788

THE GRID

One of the most impressive machines at the PC 83 Faire was the Compass Computer from GRID Systems. It is a small brief case size computer designed in the tradition of flat black stereo equipment, tasteful and sleek. The screen flips up and uncovers the keyboard. In short it has EVERYTHING but a descent price tag. It has 8086-8087 chips, 256K, 384K additional bubble memory, 320X240 bit-mapped display that is only 6 inches across, integrated modem (1200-300), an optional plug in hard disk or floppy drive in a matching case, 11 lbs. in weight, and runs MS-DOS. This is one of those products that you will no doubt see in Playboy Magazine for the ultimate in powerful computer gear that fits in your briefcase. I'd buy one in flash if it didn't cost $8,000. A preview of things to come.

INFAMOUS SOFTWARE COMPANIES

When it comes time to name your own software company it is important to pick an appropriate name. Certainly a name like SORCIM ("micros" spelled backwards) is doing well selling SuperCalc so obviously most any name will do. But there are a few names that it might be best to stay clear of. For example:

Sioux Ware Systems
Rising Sign Software
Support Ware
G. I. Systems

COMBAT NOTES

by Joe Rayhawk

This Batch file will assist you while using the BASCOM Compiler. It should be considered a follow-on to the BAT files that Peter Norton discussed for use with PASCAL in Volume 1 of Programmer's Journal.

The left justified lines below are from a file I have on my BASCOM disk called COM.BAT. When I wish to compile and link a basic routine, I place the disk it is on in my B: drive and invoke COM.BAT by: COM <filename> Note: No B:

The indented text below is comments I added for explanation.

```
REM . . Going to Compile the BASIC Program: B:%1
```

This DOS REM statement is just to remind that I have the BASCOM disk in and that my source had better be in the B: drive.

```
B:
```

Switch default disk to B: drive so that all work and output files go to the B: drive.
Aside: I keep the BASCOM and Library disks write-protected to protect me from me.

```
A:BASCOM %1 /E %2 %3 %4 ;
```

Invokes BASCOM on the file mentioned as the first argument in the invocation of COM.BAT.

I allow myself up to 3 additional parameters, but rarely use more than one more.

The /E is used on all my compiles because all my applications do some ON ERROR trapping. I limit my error-trap routines to do only "RESUME line" functions. The errors I am trapping are usually user input errors. I generally tell them what the error was (if I know) and branch back up to the input request. This covers the vast majority of the errors and keeps me from trying to write clever recovery routines which are often inscrutable and inescapable to the novice user.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

I do not include /O on the line as most of my work is for packages that CHAIN among programs. Hence, the use of BASRUN.EXE is far more practical than including the stand-alone .EXE functions by default.

Finally, the semicolon blocks all the other prompts and results in only the .OBJ file being produced.

```
A:
```

Switch back to the A: disk as a default.

```
REM . . Switch to BASIC Library Disk when Linker Pauses.
```

REMind myself why I set up LINK to pause.

```
LINK B:%1.OBJ,B:%1.EXE,,
```

Going to link the just produced .OBJ file into a .EXE file on B: drive. The two commas have two purposes. First, cause a B:%1.MAP file to be produced. I look at these occasionally. Second, the last comma causes the LINKER to pause for the Libraries request. This gives me a chance to swap from the BASCOM/LINK disk to the Library disk.

```
PAUSE . . Press ENTER to DEL B:%1.OBJ and B:%1.MAP
```

After successful LINK, I normally want to delete the intermediate files. If the LINKER fails due to space problems, then I want to keep the .OBJ file.
If I am dealing with a large program, I may want to review the .MAP file to see if I am nearing limits.

```
DEL B:%1.OBJ
```

Throw away the .OBJ; I have gotten my desired .EXE file.

```
DEL B:%1.MAP
```

Throw away the .MAP; I am not concerned about storage.

```
REM . . Just Link-Edited BASIC Program: B:%1
```

Just in case, remind me what I was doing. (Actually, to WHAT I was doing it.)

I keep duplicate copies of COM.BAT on the two Compiler disks. After the LINK step, DOS opens the COM.BAT file and gets down to the place where it left off and continues executing from there. It gets fooled into thinking the COM.BAT on the Library disk is the same one it was reading on the BASCOM disk.

Finally, If you have double-sided drives, you can fit both the BASCOM/LINK and Library files on one diskette. In which case, you can alter the REM and the LINK step. This will save one ENTER and much floppy-flipping.

COM.BAT

```
REM . . Going to Compile the BASIC Program: B:%1
B:
A:BASCOM %1 /E %2 %3 %4;
A:
REM . . Switch to BASIC Library Disk when Linker Pauses.
LINK B:%1.OBJ,B:%1.EXE,,
PAUSE . . Press ENTER to DEL B:%1.OBJ and B:%1.MAP
DEL B:%1.OBJ
DEL B:%1.MAP
REM . . Just Link-Edited BASIC Program: B:%1
```

□

Hebrew for the IBM PC

Hebrew chips should be available this month as announced by IBM Europe. Specifically the Haifa representative indicated that the monochrome and Epson printer can have their character chips replaced so that they show Hebrew as well as English. The Hebrew is displayed right to left and the English can be intermixed.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WE KNOW YOU'RE SERIOUS ABOUT THE IBM PC.

SO ARE WE!

WE KNOW WHAT YOU WANT.

- *Detailed how-to articles*
- *In-Depth product reviews*
- *Programming tips*
- *Problem reporting and solutions*
- *Code listings*

WE'VE ALWAYS HAD IT ...

Some features in previous issues were a technical look at the system unit, number crunching, utility programs, asynchronous communications, monitor and printer character sets, personalizing DOS, BASIC vs. Pascal, menu generation programs, unleashing your printer, evaluation of hard disk drives, and plotting data with PC graphics.

WE ALWAYS WILL.

Features for 1983 will include extensive information about networks, database managers, spreadsheets, operating systems, the 8087 chip, professional program development, and much more.

WE KNOW WHICH MAGAZINE YOU NEED ...

PERSONAL COMPUTER AGE

The Definitive Journal for the IBM Personal Computer User

CALL TOLL FREE: **800-824-7888 operator 77**
California only: 800-852-7777 op. 77

I NEED WHAT YOU'VE GOT! **ENTER MY SUBSCRIPTION TODAY!**

☐ I WANT INFORMATION ABOUT BACK ISSUES, TOO!

☐ PAYMENT ENCLOSED ☐ BILL ME

CHARGE MY:

☐ VISA CARD NO. _____

☐ MASTERCARD EXPIRATION DATE _____

AUTHORIZED SIGNATURE _____

☐ 1 yr. (12 issues) $24.00
☐ 2 yrs. (24 issues) $45.00
☐ 3 yrs. (36 issues) $64.00
☐ NEW ☐ RENEWAL

FOREIGN SUBSCRIPTIONS
1 YR. (12 ISSUES)
CANADA/MEXICO $29.00

OTHER COUNTRIES, WRITE FOR RATES
PAYMENTS MUST BE IN U.S. FUNDS DRAWN ON A U.S. BANK

NAME _____

ADDRESS _____

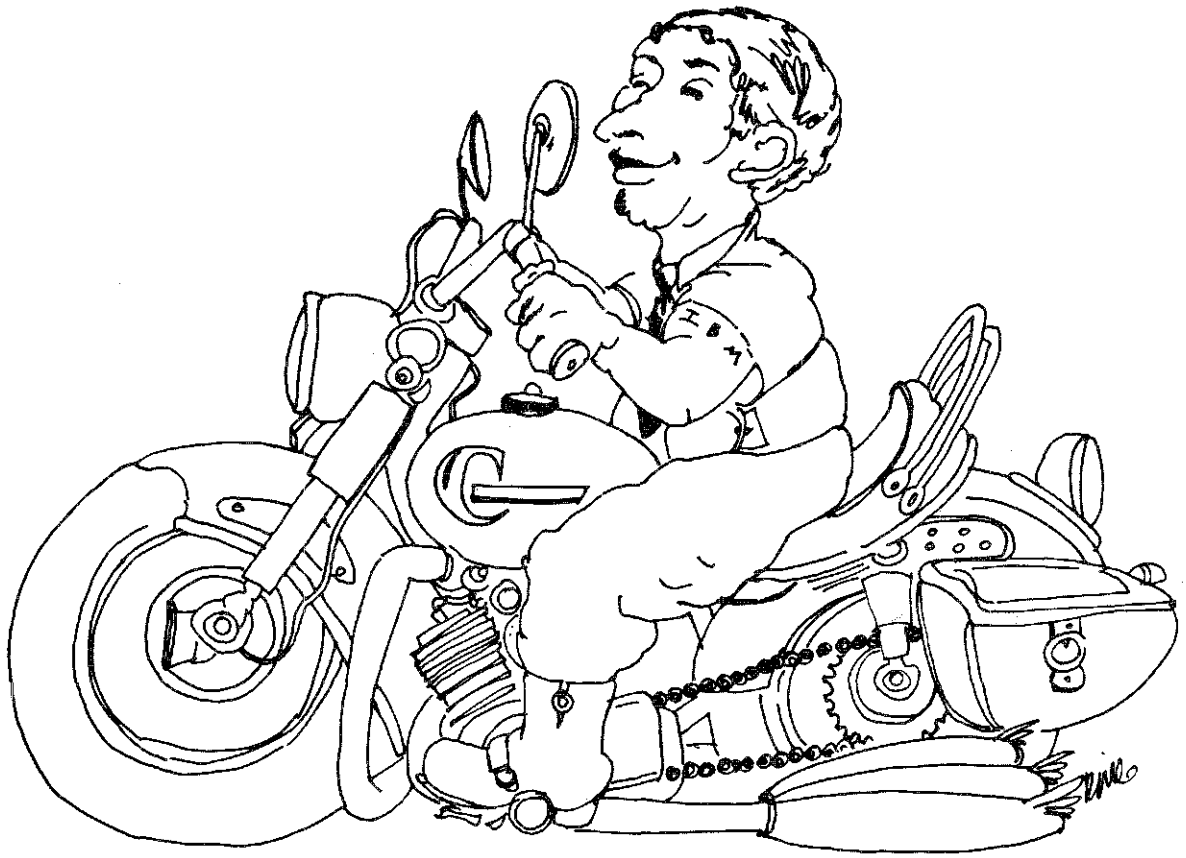
CITY _____

STATE _____ ZIP _____

Personal Computer Age

Department PJ5
P.O. Box 70725
Pasadena,
California 91107

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



BORN TO PROGRAM

PROGRAMMER'S JOURNAL

SUBSCRIPTION

Include your:

Name _____

Company _____

Address _____

City _____

State_____ Zip_____

1 year bi-monthly subscription rate:
$24 U.S. ($30 Canada, $40 other)
☐ Check enclosed
☐ Bill me

Signature_____
Begin my subscription with: ☐ First Issue ☐ Current Issue

Your system configuration:

Memory (K) 64 128 256+

Disk Drives 1 2 160 320 HD

Monitor: Color Mono

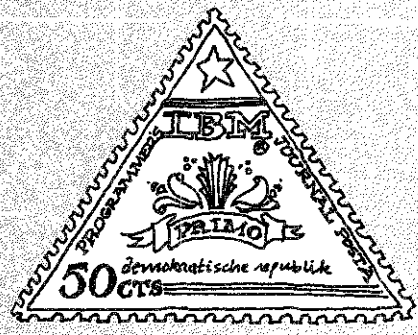
Languages you use: _____

Languages interested in: _____

Send to:
PROGRAMMER'S JOURNAL
2765 Potter St.
Eugene, OR 97405
Ph. (503) 687-0398

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



$4.00

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED